A computational model of Modern Standard Arabic verbal morphology based on generation

by Alicia González Martínez

under the supervision of Dr Antonio Moreno Sandoval

Dissertation submitted in partial fulfillment of the requirements for the degree of PhD

Madrid, December 2012

Universidad Autónoma de Madrid

Facultad de Filosofía y Letras

Departamento de Lingüística

Laboratorio de Lingüística Informática LLI-UAM

This dissertation was completed thanks to an FPU scholarship from the Universidad Autónoma de Madrid, and was carried out in the LLI-UAM Laboratory of Computational Linguistics, Department of Linguistics, under the supervision of Dr Antonio Moreno Sandoval

para Tronspmkins, que me mostró cerdos volar
para Su, que me regaló un encinar y me enseñó a hozar

"Why are you working so hard? Why don't you come and play?" But the stubborn bricklayer pig just said "no". "I shall finish my house first. It must be solid and sturdy. And then I'll come and play!"

Three little pigs

Abstract

The computational handling of non-concatenative morphologies is still a challenge in the field of natural language processing. Amongst the various areas of research, Arabic morphology stands out due to its highly complex structure. We propose a model for Arabic verbal morphology based on a root-and-pattern approach, which satisfies both computational consistency and an elegant formalization. Our model defines an abstract representation of prosodic templates and a set of intertwined morphemes that operate at different phonological levels, as well as a separate module of rewrite rules to deal with morphophonological and orthographic alterations. Our verbal system model asserts that Arabic exhibits two conjugational classes. The computational system, named Jabalín, is focused on generation—the program generates a full annotated lexicon of verbal forms, which is subsequently used to develop a morphological analyzer and generator. The input of the system consists of a lexicon of 15,452 verb lemmas of both Classical Arabic and Modern Standard Arabic—taken from El-Dahdah (1991)—comprising a total of 3,706 roots. The output of the system is a lexicon of 1,684,268 verbal inflected forms. We carried out an evaluation against a lexicon of inflected verbs provided by the analyzer ElixirFM (Smrž, 2007a; 2007b), which we considered a Golden Standard, achieving a precision of 99.52%. Additionally, we compared our lexicon with a list of the most frequent verb lemmas—including the most frequent verbs from each conjugation—taken from Buckwalter and Parkinson (2010). The list includes 825 verbs which are all included in our lexicon and passed an evaluation test with 99.27% of accuracy. Jabalín is available under a GNU license, and can be accessed and tested through an online interface, at http://elvira.lllf.uam.es/jabalin/, hosted at the LLI-UAM lab. The Jabalín interface provides different functionalities: analyze a form, generate the inflectional paradigm of a verb lemma, derive a root, show quantitative data, and explore the database, which includes data from the evaluation.

Key words: Computational Linguistics, Natural Language Processing, Arabic Computational Morphology, Root-and-Pattern Morphology, Non-concatenative Morphology, Templatic Morphology, Root-and-Prosody Morphology, Computational Prosodic Morphology.

Resumen (Spanish)

Los sistemas morfológicos de tipo no concatenativo siguen siendo uno de los mayores retos para el procesamiento del lenguaje natural. Entre las diversas líneas de investigación, el estudio de la morfología del árabe destaca por ser un sistema de gran complejidad estructural. En el presente proyecto de investigación, se propone un modelo de morfología verbal del árabe basado en un enfoque *root-and-pattern*, así como formalmente elegante y coherente desde el punto de vista computacional. El modelo propuesto se apoya fundamentalmente en una formalización abstracta de los esquemas prosódicos y su interrelación con el material morfológico. Paralelamente, el sistema cuenta con un módulo de reglas que tratan las alteraciones morfofonológicas y ortográficas del árabe. El modelo del sistema verbal propone, y se asienta en la idea de que, existen sólo dos clases conjugacionales en árabe. El sistema computacional, llamado Jabalín, está orientado a la generación: el programa genera un lexicón de formas verbales con la información lingüística asociada. El lexicón se emplea a continuación para desarrollar un analizador y generador morfológicos. Como entrada, el sistema recibe un lexicón de lemas verbales de 15.452 entradas (tomado de El-Dahdah, 1991), que combina léxico tanto del árabe clásico como del árabe estándar moderno, y cuenta con un total de 3.706 raíces. La salida es un lexicón de 1.684.268 formas verbales flexionadas. Se ha llevado a cabo una evaluación contra un lexicón de formas verbales extraído del analizador ElixirFM (Smrž, 2007a; 2007b), con una precisión de 99,52%. Por otro lado, el lexicón se ha evaluado también contra una lista de verbos más frecuentes (incluyendo los lemas más frecuentes de cada tipo de conjugación) sacada de Buckwalter y Parkinson (2010). El total de los 825 verbos que componen la lista están incluidos en nuestro lexicón de lemas verbales y presentan una precisión del 99.27%. El sistema Jabalín, desarrollado bajo licencia GNU, cuenta además con una interfaz web donde se pueden realizar consultas en árabe, http://elvira.lllf.uam.es/jabalin/, albergada en el LLI-UAM. La interfaz cuenta

con varias funcionalidades: analizar forma, generar flexión de un lema verbal, derivar raíz, mostrar datos cuantitativos, y explorar la base de datos, que incluye los datos de la evaluación.

Palabras clave: Lingüística Computacional, Procesamiento del Lenguaje Natural, Morfología Computacional del Árabe, morfología *root-and-pattern*, morfología no-concatenativa, morfología templática, morfología *root-and-prosody*, morfología prosódica computacional.

Agradecimientos

A todo cerdo le llaga su San Martín, y yo no iba a ser menos. Y es que cuando una tesis está ya terminada, el doctorando, ya fuera de sí, se empeña en seguir hozando y hozando en busca de más trufas y bellotas; "¡no es suficiente!, ¡no es suficiente!". Pero por mucho que se obceque, y por mucho que le imponga, llega el momento de terminar. Uno entonces tiene que aceptar que ya no es un lechoncillo salvaje, sino un jabalí bien lustroso y satisfecho, listo para la matanza. Y es en ese preciso momento, cuando el jabalí levanta al fin el hocico, sereno, y se para a pensar, afortunado, en todos los que acompañaron su trote y en los que le enseñaron a bellotear.

El tesinando es un animal ególatra donde los haya, a quien las largas noches trabajando en la tesis le han convertido en un ser retraído y huraño. Y mucho han tenido que sufrir y soportar los que se han mantenido a su lado. A todos ellos, que han agrantado refunfuñeos crónicos y berrinches descomunales, se lo agradezco con especial cariño.

A Santa Bárbara bendita, que en el cielo estás escrita con papel y agua bendita, por darme constancia y paciencia, que como buen totem me has estado troleando para que finalmente pudiera registar esta tesis un 4 de diciembre.

A mis profes de filología, que a pesar de mi arrebato tránsfuga en aras de la lingüística, me habéis marcado para siempre. ¡Ante todo, filóloga!

A Waleed, que aunque quizá no lo recuerde, se hizo cargo de aquella olvidada asignatura de primero, de esas que a veces los profesores no quieren dar, pero que son las que marcan a los estudiantes. Sus clases, y su dedicación a los alumnos, no se olvidan.

A Iñaqui, por sus curradísimas clases de árabe y sus simpáticas chanzas. Gracias a sus enseñanzas, me pude defender en la lengua de la Dad en mis trotes por el mundo árabe y, como los buenos estudiantes, fui el chascarillo pedante de los zocos árabes.

A Ana Ramos, de quien me acordé redactando esta tesis, por el sobrecogedor suicidio de las letras enfermas.

A mis profesoras sirias, en aquel verano que tanto aprendí en Damasco.

A Theophile, el maestro de la morfología, por ofrecerme sus valiosos consejos.

To Jirka Hana, I am grateful for the opportinuty he gave me to stay those three months in UFAL.

A Antonio, mi director de tesis, de quien nunca olvidaré su enorme paciencia durante mi etapa de formación y su inquebrantable apoyo, un privilegio del que disfrutan muy pocos tesinandos; por ofrecerme sus ideas y su experiencia, por acogerme en el laboratorio y animarme a investigar, y por darme siempre la mayor libertad.

Y no puedo olvidar todos los que han pasado por el labo, con quienes he compartido grandes momentos y cuchufletas: Ana Ledesma y su paralelo mundo de la pragmática. Marta y Manuel, que no nos han dejado tiempo para aburrirnos con sus historietas. Doaa, a quien le debo su inestimable ayuda en mis difíciles comienzos. Sus consejos y su dedicación han sido el mejor de los recibimientos en éste, el espinoso mundo de la investigación. Conchi, esa primera informática que abrió la caja de pandora. Antonio Pastor, que siempre estaba ahí para instalarme los programas que necesitara. Leo, quien ha estado todos estos años trabajando al lado, y ambos nos hemos mantenido a flote en los momentos bajos del labo. Dalia, una de esas personas que el buen recuerdo siempre conserva. El glotón de Jordi Casanova, conocido en tierras niponas como Ryo Tzutahara, ¡gran gurú de la gastronomía! Carlos, “el de japo”, que nos trae

nuevos aires a las filas de los becarios. Y, cómo no, no podría olvidarme de Azucena, que tantos días me ha hecho compañía cuando me quedaba hasta bien tarde trabajando en el labo.

A Carlos, mi maestro, que me enseñó a pensar, que alumbró el trabajo de esta tesis y cada una de sus ideas; y que siempre me ha hecho sentir de pata negra.

A Susana, que es todo lo que me gustaría ser. A ella le debo haber terminado esta tesis. Suyo es casi todo el código que le dio alas a mi modelo, y suyo es el tiempo que me regaló en tantos y tantos momentos bajos.

A Alexis, por ofrecerme siempre una sonrisa de entre ceja y ceja. Y a Laura, por agrandarla. A Cristina, por los buenos momentos y por perdonarme siempre mis vaivenes de ánimo.

A mis amigas de siempre. Ana José y sus idolatrías que marcaron nuestra adolescencia, Laura y sus admirables conocimientos porcinos (amén de otros animales), a quien siempre he tenido a un paso. Scully, con quien compartí la música por largo tiempo. Cris, que es como si hubiera estado siempre, y nunca me ha dejado a un lado. Y a Nasi, claro está, mi hermana de rabietas. No nacimos hermanas porque no hacía falta forzar nada.

A mi tía, por sus cafés, y por haber estado todos estos años pendiente de mis idas y venidas de favores en la universidad.

A mi hermano fernando, que es mi infancia, por sus inspiradores rugidos metaleros, que en los momentos de más bajón, cuando parece que uno no puede con la tesis, dan el poder necesario para continuar.

A mi hermano Carlos, mi otro gran protector, ¡y el hermano que mejor se viste del mundo! Porque nunca a nadie un hermano le ha mimado tanto. ¡Ah, y a Ana, ese lujo de cuñada!

Y por último, a los que empezaron todo esto, a mis padres, por su eterno cariño y por su inaltareble ánimo. No parece cosa de este mundo la certeza que siempre conservo de que jamás dejarán de sujetarme.

¡Bellotas para todos!

Table of contents

List of Tables

List of Figures

Transliteration Chart

Arabic letter	Name of letter	Transliteration	IPA[1]	Unicode[2]
ء	isolated hamza	c	ʔ	0621
آ	alif mamduda	Ã	ʔaː	0622
أ	hamza above alif	Á	ʔ	0623
ؤ	hamza above waw	ú	ʔ	0624
إ	hamza below alif	À	ʔ	0625
ئ	hamza above ya	ý	ʔ	0626
ا	alif	A	-	0627
اً	fatha+alif	aA	aː	064e+0627
ب	ba	b	b	0628
ة	ta marbuta	ä	t	0629
ت	ta	t	t	062a
ث	tha	þ	θ	062b
ج	jim	j	ʤ	062c
ح	Ha	H	ħ	062d
خ	kha	x	x	062e
د	dal	d	d	062f
ذ	dhal	ð	ð	0630
ر	ra	r	r	0631

[1] International Phonetic Alphabet (http://www.langsci.ucl.ac.uk/ipa/).

[2] Computing standard for encoding the writing systems of the world languages (http://www.unicode.org/).

ز	zay	z	z	0632
س	sin	s	s	0633
ش	shin	X	ʃ	0634
ص	Sin	S	sˤ	0635
ض	DaD	D	dˤ	0636
ط	Ta	T	tˤ	0637
ظ	Dha	Z	ðˤ	0638
ع	‘ayn	ç	ʕ	0639
غ	gayn	g	ɣ	063a
ف	fa	f	f	0641
ق	qaf	q	q	0642
ك	kaf	k	k	0643
ل	lam	l	l	0644
م	mim	m	m	0645
ن	nun	n	n	0646
ه	ha	h	h	0647
و	waw	w	w	0648
ـُو	damma+waw	uw	uː	064f+0648
ى	alif maqsura	Y	-	0649
ـَى	fatha+alif maqsura	Ay	aː	064e +0649
ي	ya	y	j	064a
ـِي	kasra+ya	iy	iː	0650+064a
ـَ	fatha	a	a	064e
ـُ	damma	u	u	064f
ـِ	kasra	i	i	0650

ً	double fatha	â	an	064b
ٌ	double damma	û	un	064c
ٍ	double kasra	î	in	064d
ّ	shadda	~	[gemination]	0651
ْ	sukun	·	-	0652
ـ	tatweel or kashida	¯	-	0640

TABLE 1 Transliteration chart

The aim of this transliteration is to have a one-to-one mapping with the Arabic characters, since in computational linguistics languages are usually treated through their writing systems. For this reason, we are going to use this transliteration for all the representations of Arabic forms, even to describe phonological material—we consider that using multiple systems of representation would result in unnecessary complications. The reason for developing a new transliteration system, and further explanations on the conventions adopted for it are given in appendix A.

About this work

The present study deals with the computational treatment of Arabic verbal morphology. Our aim is first to generate a lexicon of inflected verbal forms from a lexicon of lemmas based on a root-and-pattern model. Subsequently, the lexicon will be integrated in a system of verbal analysis and generation within the Jabalín project. A preliminary quantitative analysis will be performed on both lexicons.

This work owes much of the methodology and previous research developed in the LLI-UAM laboratory. Especially, it takes its inspiration from the work of Dr Antonio Moreno Sandoval in his PhD dissertation "Un modelo computacional basado en la unificación y el Análisis y Generación de la Morfología del Español" (1991), in which he presented a model or verbal morphology for the Spanish language. We have attempted to continue this line of research and applied it to Arabic. Likewise, the beginning of this work owes much of its effort to Doaa Samy, who described and developed a first prototype of an Arabic morphological analyzer in the LLI-UAM laboratory. The work she carried out was briefly outlined in her PhD dissertation, "Recursos bilingües de ingeniería lingüística para el procesamiento de español y árabe" completed in 2005 under the supervision of Prof. Moreno Sandoval, and continued shortly afterwards. She described a verbal model based on paradigms using the reference book "A Dictionary of Arabic Verb Conjugation" by El-Dahdah (1991) as a starting point and further developed a tagset to annotate verbal forms. Without this previous approximation to the complex problem of Arabic morphology, this thesis would have suffered several drawbacks due to the inexperience of the author.

The body of the thesis is organized in 7 sections. Additionally, there are 5 appendixes which contain relevant information for the thesis.

Section 1 presents the current state-of-art in this subject. It is a general overview of the Arabic language, its writing system, an introduction to Arabic phonology and morphology, and a brief summary of the sate-of-art in Arabic computational morphology.

Section 2 presents the scope and objectives of the thesis, as a result of the needs found in the field of study, which have been presented in the introduction.

Section 3 shows the materials and methods used.

Section 4 presents the results obtained in this thesis. It is divided to 4 subsections: (1) the linguistic framework developed to represent the Arabic verbal system, (2) the computational model for the generation of Arabic verbs, (3) the evaluation carried out on the lexicon of verbal forms extracted from the generation system, and (4) the Jabalín online interface, a web application for analyzing and generating Arabic verbs and deriving Arabic roots.

Section 6 discusses the main topics stated by the study and the most relevant ideas.

Section 7 enumerates the conclusions found in this work.

Appendix A contains an explanation of the criteria used for developing the transliteration system. Additionally, we make a comparison with existing ones.

Appendix B includes some quantitative data extracted from the lexicons provided by the Jabalín project and some preliminary analyses from this data.

Appendix C includes the full tagset of Arabic verbal system, along with the equivalences with ElixirFM.

Appendix D lists the codes used by the verbal generation system to classify Arabic verbs and their equivalences with both the Arabic and western conventions.

Appendix E provides the python code for all the Jabalín project—the verbal generation system, the evaluation and some additional features.

Introduction

1 The Arabic language

The term Arabic refers to a genetically related group of languages spoken nowadays mainly in the North of Africa and the Middle East. These languages are commonly known as the Arabic dialects. Additionally, the term is also applied to the formal language used in all these regions, known as Modern Standard Arabic (MSA)[3]. MSA is derived from the Arabic spoken languages and from Classical Arabic, the latter being a literary language which dates back to the 7th century. Classical Arabic plays an important role in Arab societies today for being the liturgical language of the Islamic religion and the language in which the Quran—the Holly Book—was written. Its main influence from the spoken language was the Meccan dialect, since it was the Prophet Muhammad's mother tongue (Hinds et al., 1987; Erwin, 1963; Ryding, 2005).

Thus, Arabic refers to three linguistic entities:

1. Arabic spoken varieties, the natural languages of the Arab people
2. MSA
3. Classical Arabic

Unlike dialects, MSA and Classical Arabic are not natural languages, in the sense that they do not evolve spontaneously and they do not have native speakers. The mother tongues of Arab people are rather the Arabic spoken varieties. However, the linguistic

[3] From now on and, unless otherwise specified, the terms 'Arabic' and 'MSA' will be used interchangeably to refer to MSA.

competence of Arab speakers is inevitably the main source of cognitive research for analyzing MSA and Classical Arabic. According to the web version of the Ethnologue[4], all varieties of Arabic together have approximately 206 million L1 speakers worldwide.

This work will be limited to the study of MSA, since it is the written language used *par excellence* in the Arab world. In Arabic, MSA and Classical Arabic are both referred to as اللغة الفصحى, 'the most eloquent' (Ryding, 2005).

Genetically, Arabic belongs to the Semitic Language Family, which itself is part of the Afroasiatic Phylum (Watson, 2007). The Afroasiatic phylum is divided to five families:

1. Tamazight (Berber), spoken in North Africa
2. Chad languages, in the Northwest of Africa
3. Ancient Egyptian and Coptic
4. The Cushitic languages, in the Northeast
5. the Semitic family, which includes extinct languages such as Akkadian (Assyrian and Babylonian), Canaanite and Phoenician, and living languages such as Arabic, Hebrew, Aramaic (including Syriac) and Amharic (Ryding, 2005)

Typologically, MSA is usually classified in literature as a synthetic language of the fusional type, i.e. it presents a morphological system in which the morpheme-per-word ratio tends to exceed one, and each morpheme typically codifies more than one morphological trait, which cannot be segmented. Its preferred order of sentence constituents is Verb-Subject-Object. However the Subject-Verb-Object is gaining weight in modern usage (Attia et al., 2012). It is considered a pro-drop language, i.e., subject pronouns may be dropped (Farghaly, 2009). Besides, its semantic organization of syntax reflects the typical nominative-accusative system of grammatical case. The

[4] "Ethnologue report for language code: arb", http://www.ethnologue.com Web. 27 Apr. 2012.

nominative-accusative case marking system gives the same treatment to agents of transitive constructions and agents of intransitive constructions, distinguishing them from patients of transitive events. Further, it presents a much more complex system of morphosyntactic agreement than other languages such as French or Spanish (Holes, 2004; Roth et al., 2008; Habash, 2010)[5].

The linguistic systems described above are all simultaneously in use in the Arab countries, covering different functions in society. This sociolinguistic situation is known as diglossia. In fact the term was first used with this meaning to describe the linguistic reality of Arab societies. In 1930, the Arabist William Marçais characterized the Arab countries using the term diglossie, and Charles A. Ferguson systematized this special situation in his article Diglossia, published in the Word journal in 1959 (Bassiouney, 2009).

Alan S. Kaye defines digglosia as "a situation in which two varieties of the same language live side by side, each performing a different function" (Kaye, 1990:675). He characterizes the two varieties as a 'high' variety, for formal situations, and a 'low' variety, for informal and colloquial situations. MSA and Classical Arabic cover the former, whereas the Spoken Varieties comprises the latter.

[5] A great deal of computational systems carried out in the field of natural language processing have encountered this situation (Roth et al., 2008; Habash, 2010; Alkuhlani and Habash, 2011; El Kholy and Habash, 2011).

1.1 Arabic spoken varieties اللغات العامية

Arab speaking countries spread from Northwest Africa to the Middle East, covering all the Arabian Peninsula. They comprise the so-called Arab League, with a total of 22 members.[6] Additionally, four other countries have Standard Arabic or an Arabic dialect as an official language: Chad, Eritrea, Israel and Malta.

Following the geolinguistic division of the Arab world proposed by Habash (2010:2), there are seven main groups of colloquial Arabic varieties:

1. Egypt and Sudan
2. Levantine: Lebanon, Syria, Jordan, Palestine, Israel
3. Gulf countries: Kuwait, UAE, Bahrain, Qatar, Saudi Arabic, Oman
4. North African (*maghrebi*): Morocco, Algeria, Tunisia, Mauritania, Libya
5. Iraqi Arabic: has elements of both Levantine and Gulf
6. Yemeni Arabic: is often considered in its own class
7. Maltese Arabic

With regard to their social status all instances of spoken Arabic are considered low prestige varieties—with the exception of Maltese—and consequently, as Owens (2006:9) remarks, they have no official legitimization in the Arabic world.

[6] The members of the Arab league are Algeria, Bahrain, Comoros, Djibouti, Egypt, Iraq, Jordan, Kuwait, Lebanon, Libya, Mauritania, Morocco, Oman, Palestine, Qatar, Saudi Arabic, Somalia, Sudan, Syria, Tunisia, United Arab Emirates and Yemen (Habash, 2010).

1.2 Classical Arabic فصحى التراث

Owens defines Classical Arabic as "the endpoint of a development within the complex of varieties of Old Arabic" (Owens, 2006:5). Classical Arabic was the standardized and written version of a group of spoken languages used in the Arabian Peninsula in the pre-Islamic times. These Arabic varieties spoken in the Peninsula before the rise of Islam, i.e. Old Arabic, are said to exist as early as the fourth century AD (Owens, 2006).

Classical Arabic enjoys a special status in Arab societies today, and by extension in the whole Muslim world. Amongst one of the most active religious communities all over the world, Arabic is not just a sacred language, but also the language of God himself. Hence, it is widely respected and meticulously maintained as an untouched linguistic system. MSA, its formal fellow, is less prestigious in this sense, yet it enjoys a greater prestige than the spoken varieties.

1.3 MSA فصحى العصر

We can depict MSA as a simplification of Classical Arabic, simplifying the most obscure structures of the system and reducing its overenriched vocabulary, while allowing some lexical and grammatical innovations from the spoken varieties. For instance, MSA tends to reject too complex *idafaat*[7], shows many morphological innovations and adopts many loanwords from the dialects.

Nowadays, in Arab culture, MSA covers the majority of the formal situations in society. It is the language of education; Arabs learn it in school and use it at

[7] Idafa إضافة, 'annexation' is a syntactic construction in which two or more nouns function as one phrase holding a genitive case relation (Ryding, 2005).

university, news broadcasts and mass media, and it is also widely used in literature and written language in general. Moreover, MSA performs the function of being the language of communication, a so-called *koiné*, throughout the Arab world.

1.3.1 MSA as a non-natural language

As stated above, we focused our research in the variety of Arabic known as MSA, i.e., the formal language *par excellence* used nowadays in the whole Arab world. We pointed out that this linguistic system is not a natural language, since it does not have real native speakers. The native languages of Arab people are the corresponding spoken varieties; they learn MSA through the educational system, thus in a non-natural way. This means that MSA speakers are always influenced by their native spoken variety, so they will inevitably present slant intuitions about MSA, being interfered by their mother tongues; all their perceptions about the language are affected and influenced by the structures and rules of their mother tongues.

Ferrando, among other authors[8], mentioned this problem and the fact that it is commonly ignored in studies in the field, especially under the generative scope. As Ferrando stated, it is important to bear in mind that the competence of Arab speakers is not reliable. "(...) la lengua árabe estándar no dispone de hablantes nativos en el sentido estricto del término, puesto que es una lengua de cultura que no se aprende sino a través del estudio, es decir, de forma no natural. Eso provoca que los juicios y las intuiciones de los arabófonos (...) carezcan de la competencia necesaria por verse a menudo interferidos por sus lenguas maternas, los dialectos neoárabes"[9] (Ferrando,

[8] Ferrando mentioned the article "Formal vs. Informal Arabic: Diglossia, Triglossia, Tetraglossia, etc., Polyglossia viewed as a continuum" from Kaye, 1994.

[9] "(...) MSA do not have native speakers in the strict sense of the term, as it is a cultural language only learned by means of study, i.e. in a non-natural way. Therefore, the judgments and the intuitions of the speakers (...) lack of enough competence, as they are often interfered by their mother tongues—the Neoarabic dialects".

1999:12). Holes presents a similar view: "The problem in describing the phonology of MSA is that MSA is not natively spoken by any group of Arabs, so no contemporary regional or social grouping can reasonably claim that its habits of performance represent a model of what is *correct*" (Holes, 2004:56).

Additionally, the fact that MSA is not a natural language makes us wonder about its nature and relation with spoken languages. We said that MSA is created from the old 'Arabic dialects', yet until now there is not a truly precise definition of the sources of MSA; which 'old dialect' has influenced MSA the most?; which other 'dialects' are also responsible for the formation of MSA?; to what extent have the Aramaic languages—the *lingua franca* in the Middle East before the rise of Islam—and the Syriac language affected the old 'dialects' and by extension MSA?[10] Undoubtedly, a better understanding of the historical development of old and new 'Arabic dialects' and MSA, will lead us to a better comprehension of the linguistic properties of these systems. The relation between diachronic and synchronic analysis is nevertheless bidirectional: diachronic processes attested in old stages of the language may clarify the nature of inexplicable traits seen in a synchronic perspective—which may be the result of fossilized processes, no more active in the language—; inversely, deviant forms found in a synchronic view may be proof of the existence of productive operations present in an early stage of the language.

In the same fashion, there has been some discussion about the possible artificial nature of Classical Arabic and MSA linguistic structure, as it seems to be 'too perfect' when comparing it with natural systems. Kaye stated in this respect: "Arabic sticks out like a sore thumb in comparative Semitic linguistics because it's almost (too perfect) algebraic-looking grammar, i.e. root and pattern morphology" (Kaye, 1990:665).

[10] In this respect, Versteegh mentioned that "In general, substratal influence on the Arabic dialects has been invoked in many cases without much justification" (Versteegh, 2001, p.105)

1.3.2 Traditional Arabic lexicography

The science of grammar emerged in the Arab culture under the auspices of the Quranic exegesis. Yet, and contrary to common belief, it was not the main source for supporting grammatical analysis. It should be carefully observed the remark stated by Owens (2006) who claims that the main basis of Sibawayhi's analysis was not the variety of the Quranic text. The Quran was indeed a reliable source for analysis foundation, but so was pre-Islamic poetry and, more significantly, the speech of the Bedouins. The testimonies of the Arabs from nomadic societies were taken as correct and 'pure' Arabic—and thus established as authoritative and normative Arabic, in need of preservation—while at the same time the urban manifestations of the language were considered a mere 'corruption' (Versteegh, 2001); hence all the peculiarities and traits of the Bedouin speech were considered part of an homogeneous language. Each and every one of the characteristics found in the language of any Arab tribe was thus collected and incorporated into the system. In doing so, the Arabic lexicon was overwhelmingly enlarged and all the features found in that lexicon were expected to have an explanation in a ' falsely uniform' system, i.e., the standard Arabic language was in fact a collection of varieties, inevitably overlapped at all linguistic levels.

In the article *El plural fracto en semítico: nuevas perspectivas*, Ferrando (1999) provides a critical evaluation of the analysis carried out by Ratcliffe about the Semitic broken plural—embodied in his book "The 'broken' plural problem in Arabic and comparative Semitic" (Ratcliffe, 1998)—. In his analysis, Ratcliffe tries to find a morphological explanation for each and every lexical item present in the MSA lexicon. Ferrando pointed out the erroneous assumption implicit in Ratcliffe's analysis by which MSA is supposed to be an homogeneous linguistic system; on the contrary and as we have seen, at least in the first stages of its formation, MSA was based on diversity and heterogeneity as a general rule (Ferrando, 1999). In the words of Ferrando: "(...) la razón más importante de la presencia de plurales *redundantes* en los léxicos árabes es

la variación dialectal que las fuentes no discriminan, sino que más bien acogen en silencio"[11] (Ferrando, 1999:13).

Danks, in a recent work, emphasizes this idea and warns of the limitations of the studies in this field: "(…) the synchronic study is also realistically unattainable for practical reasons: any language, specially a major international variety like MSA, is in use over a geographically widespread area by speakers with a wide range of ages and social background. (…) I will therefore treat a language as if it is a self-consistent, homogeneous system, but will not be unduly surprised if a small number of anomalies defies synchronic explanation" (Danks, 2011:7-8).

In a nutshell, there are two main traits which must be carefully considered while developing an analytical representation of the MSA linguistic structures:

1. MSA is not a natural language, hence its linguistic system will present an extreme—and we may add artificial—tendency to regularity.

2. Paradoxically, as a result of the unification of diverse varieties in a standard version of the language, MSA will present 'false' redundancies and ambiguities in its lexicon, which perhaps should be analyzed as irregular items due to the heterogeneity of the system—and thus they may not 'break' the systematicity of the system principles.

The second problem is more typical of nominal morphology, which is more permissive in terms of morphological acceptability of the lexicon, than compared to verbs.

[11] "(…) the main reason for the presence of redundant 'plurals' in the Arabic lexicons is the dialectal variation that the sources do not discriminate, but rather include in silence"

2 Writing system

2.1 The Arabic script

Arabic writing system always represents consonants. Additionally short vowel characters, which consist of diacritics written above consonants, may be included, but they are seldom used in handwriting or print style. Conversely, long vowels are always written. This is because they are represented by a short vowel followed by a pure letter, i.e. a consonant, which is always present. The letters used for the long vowels *uw* and *iy* are the short vowels *u* and *i* followed by the corresponding semiconsonant *w* or *y*. In the case of *aA*, the letter which is always present is the *alif* *A* (Abu-Chacra, 2007). In old stages of the language, the *alif* represented the glottal stop phoneme. In current writing, the *alif* has lost the status of consonant, except in some context where it still represents the glottal stop—an example of this is given on page 33.

On the other hand, the gemination of consonants is represented by a diacritic symbol called *shadda* '~'. Contrary to vowels, which are optional, the shadda should always be written. Yet it is commonly absent both in print and handwriting.

This consonant-based writing system, typical of Semitic languages, is known as consonantary or *abjad*—in Arabic أبجدية عربية—and is harmoniously bound with the nature of Semitic morphology: the vocalic string of an Arabic word is generally reflecting grammatical properties, and consequently it should be inferred from the linguistic context.

The Arabic consonantary was taken from the Nabatean scrip. The Nabateans, in turn, borrowed it from Aramaic and added some letters corresponding to phonemes the Nabatean speakers lack.

This consonantary adapted from the Nabatean script was highly ambiguous notwithstanding. There were groups of different phonemes which shared the same letter form. On the other hand, the absence of vocalic symbols, though not crucial for understanding, added much ambiguity to the system (Shalaan et al., 2010). In Aramaic, dots were sometimes added below and above the letters to distinguish consonants with identical shapes. In Arabic, this habit was systematized, and dots—sometimes referred to as consonantal diacritics—in Arabic إعجام *içjaAm*—became obligatory components of the letter shapes. As for the short vowels, diacritical marks were borrowed from Syriac. Other diacritics used in the language are (Abu-Chacra, 2007; Wright, 2007; Cowan, 1958):

1. the elongation mark known as *shadda* '~'—to which we referred above,
2. *nunation*, which consists of writing two short vowels together. It is not pronounced as a long vowel, but the vowel followed by 'n'. The 'n' corresponds to the indefinite article suffix; the previous vowel is the case vowel, e.g.:

	Arabic	transliteration	pronunciation
Nominative indefinite article	بَيْتٌ	bay·tû	baytun
Accusative indefinite article[12]	بَيْتًا	bay·tâA	baytan
Genitive indefinite article	بَيْتٍ	bay·tî	baytin

TABLE 2 Indefinite case

[12] The accusative indefinite article adds a final alif as an orthographic convention.

3. *dagger alif* (الألف الخنجرية), small alif or superscript alif (Habash, 2007). It appears in archaic spellings and it is also maintained in religious terms e.g. الرَّحْمٰن, الله

This desire to maximize the disambiguation of the writing system which emerged in the Arab culture is due to the necessity of finding an unequivocal written representation of the Islamic revelation. The Quranic exegesis demanded to work with a non ambiguous text, at least with respect to orthography. And it lasted for some time, as it was not until the eighth century that complete versions of the Quran were produced (Owens, 2006). Indeed, a similar effort was carried out in the field of linguistics and grammar studies. The Arabic linguistic tradition emerged and was developed under the scope of the religious science.

It may be interesting to think for a moment about the consideration the Arab scholars gave to the writing system and its relation with the language itself. In Arabic, the typical word for *letter* is حرف *harf*, understood as each of the symbols used in the 'Arabic alphabet'—the consonants—but curiously this same word has been often used in literature to refer to other concepts. Arabic particles—meaning roughly function words—are also referred to as *harf*. In the same fashion, *harf* can be used in the sense of *word*, or even *variant of speech*, referring to dialectical variants of lexical items (Al-Nassir, 1993). Therefore, there is not a clear distinction between linguistic properties and the concept of writing, which is purely conversional and extralinguistic.

The Arabic writing system follows a right-to-left direction, does not have capitalization and is in cursive style, either in printed form or handwriting script. Each letter—i.e. each consonant—has four different shapes depending on the position in the word: start form, medial form, final form, and isolated form (Abu-Chacra, 2007). Depending on the letter the different shapes of the set will be to some degree different from each other.

Introduction

(1)

final	medial	inicial	isolated
ـه	ـهـ	هـ	ه

One peculiarity of Arabic script, which is a consequence of the cursive style, is the possibility of changing the shapes of groups of consonants when written together. The resulting character always follows expected pattern styles.

(2)

ل + ح ← لح ← لـح

There exists a stylistic tradition of lengthening the junction line which connect two consonants in order to justify paragraphs; this custom is known as *elongation*—تطويل *tatweel* or كشيدة *kashida*.

(3)

لح ← لـــــــــــــــح

2.2 The *hamza* problem

The letter *hamza* has six representations in the Arabic alphabet, so it corresponds to six different characters. The reason for this unusual ambiguity in the Arabic writing system—a phoneme corresponding to various letters—has a historical background.

At the time the Quran was being revealed and written[13], the glottal stop phoneme had disappeared in the dialect of the Hijaz. However, in the prophet's variety this sound was maintained, so the Hijazi scribes had to devise a way of representing it in the writing system. They chose the long vowels *waw*, *ya* and *alif*, for in their own variety the hamza had been replaced by these sounds. This obviously led to an ambiguity problem between the long vowels and the glottal stop (Versteegh, 1997).

Al Khalil ibn Ahmad Al Farahidi, a famous Arab lexicographer from the 8th century is said to have invented the graphic symbol of the *hamza* when he was trying to eliminate the ambiguity the sound presented in writing. He erroneously believed that the place of articulation of the glottal stop was the same as that of the letter *'ayn*, the pharyngeal fricative, so he took the upper part of the letter *'ayn*'s shape to represent the *hamza*—the glottal stop sound. Then, he simply stipulated that the *hamza* should be written above the long vowels (Al-Nassir, 1993). This is why in contemporary

[13] During Muhammad's life (ca. 570-632), there were written fragments of the Islamic revelation, but according to tradition, a complete version of the Quranic text was not available until the third Rashidun Caliph's rule (644-656)—following the Sunni tradition—Othman, who commanded the elaboration of an authoritative codex. This authoritative version of the complete Quran was not widely accepted as canonical until the second century of the Hijra, i.e., the beginning of the Muslim era and calendar (Versteegh, 2001). Hence, a standard and unified version of the Quran took more than a century to be available in the Muslim culture.

script there are different characters to represent the hamza, depending on the vocalic context:

1.	*hamza* above an *alif*	أ	Á
2.	*hamza* below an *alif* [14]	إ	À
3.	*hamza* above *waw*	ؤ	ú
4.	*hamza* above a dotless *ya'*	ئ	ý
5.	isolated *hamza* [15]	ء	c
6.	*hamza* plus *fatha* plus *alif* [16]	آ	Ã

Of course, by doing this Al Khalil did not finish entirely with the ambiguity problem, but provided another dimension to it: all five characters listed above are different representations of the same phoneme. This means that some orthographic rules are needed to represent Arabic words in full orthography. These rules will have to reflect the phonetic context surrounding the *hamza*.

2.3 Computational treatment of the Arabic script

A computational representation of the linguistic nature of MSA will inevitably have to pass through the idiosyncrasies of orthography. Consequently, we must be aware of the distinction between linguistic and orthographic characteristics, despite working with them both at the same time.

[14] When the following character is *kasra*.

[15] Isolated *hamzas* are found in modern orthography. They lack any supporting letter and are written alone.

[16] This merged character avoids the representation of two consecutive *alifs* اأ *ÁaA*, which is considered unsightly.

With regards to computational support, the majority of Arabic script idiosyncrasies—positional shape of letters, ligatures and writing direction—are automatically handled by computational applications, because they are abstracted away by those applications. The *tatweel* or elongation mark, which corresponds to Unicode U+0640, adds noise when processing texts in Arabic script, so it must be removed in a normalization preprocessing, together with punctuation marks (Goweder and De Roeck, 2001).

With respect to codification, most platforms support Arabic. Unicode/UTF-8 has become a widely accepted standard when dealing with Arabic script. However, sometimes Unicode fails to interact properly[17] with systems and turns to be an important obstacle to deal with specific applications. Apart from that, the different directionality of Arabic script is commonly a huge difficulty too. When mixing Arabic letters with Latin script, the right-to-left direction of Arabic is combined with left-to-right direction. Internally, text direction may be perfectly treated by a system—this is the least expected—but the display input is usually very complex, and inevitably fails to present a clear representation of the combined letters. For example, the display of regular expressions using Arabic script is extremely opaque in the majority of text editors.

Below, we show an example of the disordered representation of Arabic characters inside a regular expression. A transliteration is given so that the sequence of characters can be seen as they should be organized.

(4)

Arabic	(.[ً-ي])ً[و-ي](ً?ّ[ء-ي])
Transliteration	(.[c-y]~?a)[wy]a([c-y]a)

[17] One example of this is the constant failures provoked by the BOM character of the Unicode.

3 Phonology

In the following lines, we are going to examine the most relevant characteristics of Arabic phonology (Newman, 2002; Ryding, 2005; Holes, 2004; Watson, 2007; Abu-Chacra, 2007). We must keep in mind that, for the sake of simplicity, we are going to continue to use our transliteration, even for the phonological argumentations; thus phonological features are going to be related to letters themselves, without using the symbols from the International Phonetic Alphabet (IPA). This is because computational applications always require working with written texts, and although linguistic and orthographic properties must be distinguished and separated, we indeed have to deal with both of them. As a result of this, we find it more logical to generalize the use of the transliteration to all the parts of the analysis.

The next two tables show the standard places and manners of articulation of the vowels and consonants in MSA.

	Front	Central	Back
High	i ِ / iy ِي		u ُ / uw ُو
Mid			
Low		a َ / aA ٰ	

TABLE 3 Arabic vowels

The data in table 3 is taken from Ryding (2005). Each distinctive vocalic quality includes two different phonemes: a short vowel and its counterpart long vowel. Arabic vocalic system admits two diphthongs: 'وَ' *aw* and 'يَ' *ay*.

		Stop	Fricative	Affricate	Nasal	Trill	Approximant	Lateral
Labial	**Bilabial**	b ب			m م		w و	
	Labiodental		f ف					
Coronal	**Dental**		þ ث ð ذ Z ظ					
	Alveolar	t ت T ط d د D ض	s س z ز S ص		n ن	r ر		l ل
	Palato-Alveolar		X ش	j ج				
Dorsal	**Palatal**							y ي
	Velar	k ك		x خ g غ			w و	
	Uvular	q ق						
Radical	**Pharyngeal**	T ط D ض	S ص Z ظ				H ح ç ع	
Glottal	**Glottal/ Laryngeal**	c ء	h ه					

TABLE 4 Arabic consonants

The data in table 4 are taken from Ryding (2005), Holes (2004), Watson (2007), Bin-Muqbil (2006) and the International Phonetic Association (1999). The glottal stop adopts the spelling ء *c*, as represented in the table, but also the alternative spellings أ *Á*, إ *Â*, ئ *ý* and ؤ *ú*, as well as آ *Ã*, which consists of a glottal stop followed by a long vowel *a*. The letter ة *ä*—which is pronounced only if not prior to a pausal stop in the pronunciation—corresponds to an alveolar stop, so it is pronounced as ت *t*. The places

and manners of articulation of some of the phonemes in MSA may vary depending on the vernacular *dialect* of the Arab speakers[18].

The letters ص *S*, ط *T*, ض *D* and ظ *Z*, traditionally called *emphatic consonants*, تفخيم, are often described as representing pharyngealized consonants (Gussenhoven & Jacobs, 1998). Indeed, they are just consonants with two places of articulation, interdental or alveolar—depending on the cases—and pharyngeal. That is why we have located them in two different places within the table. Acoustically, they are characterized by an impression of hollow resonance. Additionally, they tend to emphasize adjacent consonants and lowered contiguous vowels (Holes, 2004). The emphatic class is typical of Semitic languages and they may present a special behaviour in some contexts. For instance, in Akkadian, roots[19] cannot contain two emphatics. For roots containing two emphatic consonants in other Semitic languages, it is expected that Akkadian dissimilate one of them according to the feature *emphatic* (Geers, 1945). As we will see, this special feature is also exhibited in Arabic.

In relation to place of articulation, Semitic languages distinguish a so-called guttural class from the rest of consonants; uvulars or velar affricates خ *x* and غ *g*, pharyngeal approximants ح *H* and ع ç, and lagyngeal stop ء *c* and fricative ه *h* are considered gutturals (Versteegh, 2008; McCarthy, 1994). Articulatorically, they are characterized by a constriction in the laryngeo-pharyngeal region (Watson, 2007) or velar and postvelar (Holes, 2004). Guttural consonants are said to constitute a natural class amongst Arabic phonemes. They show a tendency of lowering the adjacent vowels in some contexts. It has been reported that "nonemphatics /r/ /x/ /g/ /q/ also exercise a backing effect on contiguous vowels similar to that of the emphatics." Some linguists therefore propose that emphasis is "a prosodic or suprasegmental feature whose minimal domain is a consonant and an adjacent vowel" (Holes, 2004:58). In Arabic,

[18] This might explain why the bibliographical sources present mismatched data.

[19] In Semitic linguistics, the root is a discontinuous morpheme, of abstract nature, which bears semantic content. It is widely treated in the Introduction-4.4.2.1.

guttural consonants tend to induce their lowering effect on the surrounding context (Bin-Muqbil, 2006; McCarthy, 1994)—as well as in other Semitic languages, such as in Somali (Gabbard, 2010) or Ugaritic (Huehne, 2008)—sometimes motivating morphological categories, for instance the verbal pattern Iaa in Arabic, which is treated on page 75. It has long been said that the *fatha*—vowel *a*—of imperfective verbs of pattern Iaa is caused by the presence of a guttural consonant in the second or third position of the root (McCarthy, 1994; Ma'sud, 2001, mentioning Sibawayhi and Ibn Jinni).

Another interesting class is the coronal one, which includes the dental, alveolar and palato-alveolar consonants[20]: ث *þ*, ذ *ð*, ظ *Z*, ن *n*, ت *t*, ط *T*, د *d*, ض *D*, س *s*, ص *S*, ش *X*, ز *z*, ل *l* and ر *r*. The phonemes corresponding to *l*, *r* and *n* are called Coronal Sonorants; the rest, coronal obstruents. There is a systematic assimilation process affecting consonants marked by the feature coronal. The definite article prefix ل *l* is assimilated by the first consonant of words beginning with a coronal consonant. As a result of the assimilation of the *l*, the coronal consonant is doubled.

(5)	Arabic	ال + شَمس = الشَّمس
	Transliteration[21]	Al + Xam·s = AlX~am·s
	Pronunciation	caXXams

Apart from gutturals and coronals, there are two other natural classes encompassing the rest of the consonants; labial consonants include ب *b*, ف *f*, م *m* and و *w*—which is a co-articulated consonant, as well as the emphatic set; dorsals include velars and uvulars ك *k*, ق *q*, خ *x* and غ *g*, together with semiconsonants و *w* and ي *y*.

[20] In spite of being pronounced in the palato-alveolar region, the affricate ج *j* is excluded from this category. This is because it was a voiced velar stop *g* in Proto-Semitic (Versteegh, 2001).

[21] Conventionally, as is seen in *AlX~am·s*, the *l* of the article does not have *sukun*, even though it is always unvocalized—perhaps for this reason.

Introduction

MSA allows the gemination of consonants. This process is known in Arabic as تشديد *tashdid*, meaning *intensification*, and is orthographically represented by the diacritic ّ ~, called in Arabic شدّة *shadda*. In Arabic, gemination of consonants has three causes:

1. It may me lexical, i.e. the result of a root with two identical segments, as in رَدَّ *rad~a*, whose root is ردد $\sqrt{rdd}$[22].
2. It may be a derivational process which shows intensification or causativity of the act expressed by the verb, e.g. دَرَّسَ *dr~s*, meaning 'to make study' or 'to teach', derived from the verb دَرَسَ *darasa*, meaning 'to study'.
3. It may be the result of an assimilation process, irrelevant in a morphological level, as in the change: اِظْتَنَّ *AiZtan~a* > اِظَّنَّ *AiZ~an~a*, meaning 'to think', and which has a root $\sqrt{Znn}$[23].

Gemination as a derivational process is a very productive operation in the Arabic system. In the same fashion, Arabic frequently exhibits vowel lengthening processes of derivational nature. In fact, both operations are widely applied derivational processes in the Arabic verbal system. In this sense, both changes can be described as lengthening operations affecting the subsyllabic prosodic unit, i.e., the mora. In terms of moraic structure, both consonant gemination and vowel lengthening, as a derivational operation, are quantitatively identical (Danks, 2011). This is the reason why, from a computational perspective, some authors consider both operations lengthening changes (e.g.: Beesley, 1998b; Habash, 2010). Orthographically, a geminated consonant is represented by the consonant itself plus the shadda diacritic ّ ~. In turn, long vowels *u* and *i* are depicted as the graphic symbol of the short vowel plus the corresponding semiconsonant letter, thus for وُ *uw* and يِ *iy*. Long *a* is written with a short *a* plus an alif اَ *aA*.

[22] From now on, roots will be indicated with the mathematical symbol $\sqrt{}$.

[23] Note that the gemination of the *n* has lexical status, not phonological.

(6)	consonant gemination	بّ	*b~*
	vowel lengthening	ـُو	*uw*

In this sense, the *shadda* may be seen as a *consonant lengthener*, and the letters *waw* و *w*, *ya* ي *y*, and *alif* ا *A*, by extension, as *vowel lengtheners* when a short vowel precedes them.

3.1 Pronunciation styles in MSA

MSA exhibits a system of morphological case in the form of suffixes to words[24]. These suffixes present the peculiarity that they may not be pronounced in some contexts:

1. In formal style it is expected to pronounce all of them, except at the end of sentences or when a discursive pause—known as *waqf* وقف (Al-Ani, 2007)—is required.
2. In a more informal style all case endings may be ignored.

(7)	Arabic	قَالَ الطِّفلُ إِنَّهُ يَشعَرُ بِسَعَادَةٍ
	Transliteration:	qaAla alT~iflu Àin~ahu yaXçuru bisaçaAdaäî
	Full-style	qaAl-**a** aT-Tifl-**u** Àin~ahu yaXçur-**u** bisaçaAda(ä)[25]
	Pausal-style (less formal style)	qaAl alT~ifl Àin~ahu yaXçur bisaçaAda(ä)

As a consequence of this, an Arabic word has a different syllabic structure depending on the pronunciation style the speaker follows. For instance, the word قَالَ qaAla, from

[24] The suffixes are: -u for nominative definite, -a for accusative definite, -i for genitive definite, -un for nominative indefinite, -an for accusative indefinite, -in for geninite indefinite. In Table 2 we can see an example with indefinite sufixes.

[25] When case-ending is not pronounced, letter ة ä is never pronounced.

the example above, will have two syllables in full-style pronunciation, but only one syllable in pausal-style pronunciation.

Full-style	qaA.la	CVV.CV[26]
Pausal-style	qaAl	CVVC

The position of the discursive pauses depends on the places the speaker chooses to make those pauses, thus they may be quite random in some cases, and the resyllabification process may be *a priori* unexpected.

Historically, there is no clear evidence from old stages of the language which allows us to know which style pronunciation was the original one; either the full-style of the pausal-style could be a development from the other (Owens, 2006).

[26] We follow this convention of represent syllables: C: consonant; V: vowel; VV indicates a long vowel; a dot indicates a syllable bounding.

3.2 Phonotactics

Phonotactics is the discipline that studies how the phonemes of a language are combined with each other within words. One of its objectives is to determine the distribution of sound sequences which are prohibited in the language, and to formalize restrictions which cause them (Roca and Johnson, 1999).

3.2.1 Syllabic structure

Arabic syllabic structure (rev. in Kouloughli, 1986; Ryding, 2005; Watson, 2007; Wright, 2007) is highly restrictive; only a very small set of syllables are allowed. We must distinguish possible syllables in full-style pronunciation from possible syllables in pausal-style pronunciation.

In full-style pronunciation:

CV	(8)	سَبَبْ	sa.ba.bu	**CV.CV.CV**
CVC	(9)	مُستَقبَلْ	mus.taq.ba.lu	**CVC.CVC**.CV.CV
CVV	(10)	نَامَ	naA.ma	**CVV**.CV
*CVVC[27]	(11)	اِورَاقّ	Ă[28]iw.raAq.~a	CVC.**CVVC**.CV

In pausal-style pronunciation (so all of them are found at word final position):

[27] This syllable is extremely rare in full-style pronunciation. In many cases it is *resyllabized* and turned into CVC. E.g. the form جَادْتُ* **jaAdtu*—from the verb جاد *jaAd*—is reinterpreted as جَدْتُ *jadtu*. Thus, it is not going to be counted as an allowed syllable in full-style pronunciation.

[28] Although orthographically there is an alif *A*, a hamza is pronounced *Ȧ*, i.e., the word begins indeed with a consonant.

CVC	(12)	مَكتَب	mak.tab	CVC.CVC
CVCC	(13)	حَرف	Harf	CVCC
CVVC	(14)	وَزِير	wa.ziyr	CV.CVVC
*CVVCC[29]	(15)	حَاجّ	HaAj~	CVVCC

Furthermore, two global phonological rules govern the phonotactics of MSA. On the one hand, a syllable cannot begin with a vowel; on the other, a syllable cannot begin with a cluster of consonants. Additionally, if we take into account the full-style pronunciation, a syllable cannot either finish in a cluster of consonants. The formulation of these rules, considering the full-style pronunciation[30] is:

Rule 1 All syllables must have an onset

Rule 2 Consonant clusters within syllables are prohibited

To sum up, this is the possible structure of syllables in MSA in full-style pronunciation [31]:

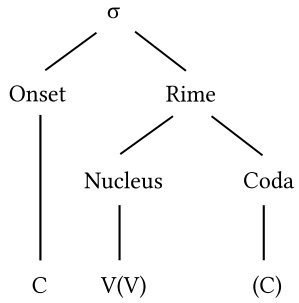


FIGURE 1 General structure of Arabic syllables

[29] This syllable is in fact pronounced as CVVC.

[30] As the present work deals with full-vocalized Arabic, we will focus our analysis on full-style pronunciation from now on.

[31] σ: syllable; μ: mora.

Prosodically, MSA discriminates light from heavy syllables—giving the same consideration to all kinds of heavy syllables. This means that CVV and CVC are treated alike in terms of length, and by extension, under the scope of prosody. As it will be shown later, Arabic words corresponding to the same morphological category will show identical or at least similar syllabic structures with regard to length. Consequently, we will represent them in terms of moras. A mora is basically a unit of timing. In words of Lagefoged and Johnson (2001:251), "each mora takes about the same length of time to say". Different syllabic lengths—determined only by the structure of the rime (Roca and Johnson, 1999)—will entail a different amount of moras:

A **light syllable** weighs 1 mora μ

A **heavy syllable** weighs 2 moras μ μ

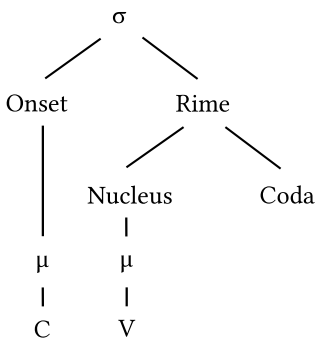


FIGURE 2 Structure of light syllables

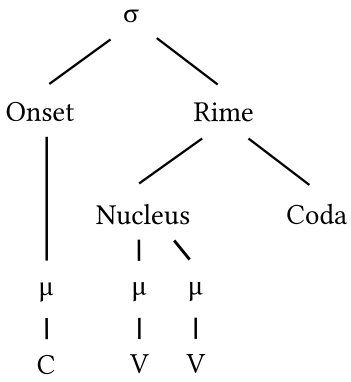


FIGURE 3 Structure of heavy syllables with long vowel

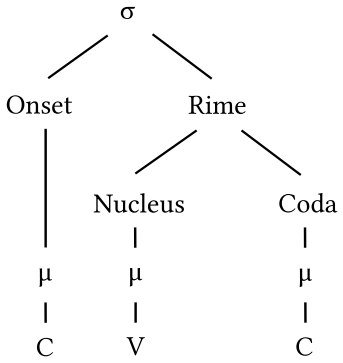


FIGURE 4 Structure of heavy syllables with consonantal coda

The rhythmic uniformity encountered in the lexical items of MSA, is in turn found in larger units of connected speech. Medieval Arab scholars noticed the importance of analyzing the suprasegmental segments of Arabic. Al Khalil, the acclaimed Arab scholar considered the father of lexicography, described and systematized the metrical system of Arabic poetry, analyzing the utterances under the scope of quantitative prosody. He defined sixteen different metrical patterns (Wright, 2007) according to their syllabic structure—classified by moraic units. His classification of the syllables is

entirely based on orthography, yet it follows linguistic principles[32]. As we previously stated, in Arabic linguistic tradition the concept of orthography was intermingled with linguistic analysis.

Khashan (2003) made an overview of Al Khalil's quantitative analysis of prosody. Al Khalil established a basic distinction in the orthographic units of writing strings; letters could be of two kinds:

a. حرف ساكن *harf sakin* (s) 'static letter', i.e. an unvocalized letter. A letter *sakin* is a letter followed by *sukun*. Letters *waw*, *ya* and *alif* are also treated as *sakin* letters when they are used as long vowels—known in Arabic as ممدود *mamdood*, i.e. letters which support neither a sukun nor a short vowel.

b. حرف متحرك *harf mutaharrik* (m) 'moving letter', i.e. a vocalized letter. A *mutaharrik* letter is a letter followed by a diacritic vowel.

The fundamental principle of this analysis is that a *mutaharrik* letter is heavier than a *sakin*. To represent this, they are marked with different weight symbols. In our transliteration, we have chosen to use '0' for *harf sakin* and '1' for *harf mutaharrik*—in Arabic, following Khashan (2003) conventions, we will use a hamza ٥ for *harf sakin* and the numeral ١ for *harf mutaharrik*.

[32] Remember that the writing system was considered part of the language, so the graphic symbols of the system present a rather accurate relationship with the language itself.

Representation in Arabic		Representation in our transliteration	
(س) ساكن	ه	Sakin (s)	0
(م) متحرك	١	Mutaharrik (m)	1

(16)	word	مَكْتُوبْ	mak·tuwbu
	Segmentation in letters	مَ + كْ + تُ + و + بْ	ma . k· . tu . w . bu
	Classification	م + س + م + س + م	m + s + m + s + m
	Weights symbols	١ + ه + ١ + ه + ١	1 + 0 + 1 + 0 + 1

The different syllabic structures are unravelled by assigning accumulative weights. This means that some sets of letters are considered—prosodically inseparable—syllabic units under the conception of Arab scholars. These units may be monosyllabic or disyllabic[33]. The accumulative weights reflecting syllabic units are mathematical-like:

Arabic	explanation
٢ = ه + ١	The sequence 1 - 0 is considered a unit with value 2
٣ = ٢ + ١	The sequence 1 - 2 is considered a unit with value 3
٤ = ٢ + ٢	The sequence 2 - 2 is considered a unit with value 4

For instance, this is the computation of the accumulative weights of the previous example:

[33] Al Khalil laid the foundations of the classical Science of Prosody by establishing a quantitative classification of prosodic feet based on different syllabic structures, i.e. sets of these syllabic units. The combination of these feet is the basic principle for the analysis of the metric system of Arabic verse. Briefly, syllabic structures lie in three general categories: سبب *sabab*, biliteral forms corresponding to syllabic structures CVC, CVV or CVCV; وتد *watad*, triliteral forms of structure CV CVC, CV CVV, CVC CV or CVV CV; and فاصلة *fasila*, quadriliteral or quintiliteral forms CV CV CVC, CV CV CVV, CV CV CV CVC, CV CV CV CV CVV (Al-Nassir, 1993).

(17)	word	مَكْتُوبٌ	mak·tuwbu
	Segmentation in letters	مَ + كْ + تُ + و + بٌ	ma - k· - tu - w - bu
	Weights symbols	١ + ه + ١ + ه + ١	1 - 0 - 1 - 0 - 1
	Accumulative weights	١ + ٢ + ٢	2 - 2 - 1
	Total weight of lexical item	٥	5

The different weights correspond to different sets of syllable combinations:

Weight symbols	Accumulative weight	Sequence of syllables
1	1	light
1 + 0	2	heavy
1 + 1 + 0	3	light + heavy
1 + 0 + 1 + 0	4	heavy + heavy

TABLE 5 Types of weights according to classical quantitative prosody

As can be seen, the super-heavy syllable CVVC, found in few rare contexts, is ignored by the traditional system of weight counting.

Interestingly, following this traditional view Khashan captures the principle expressed previously in rules 1 and 2 by means of orthography: لا يبدأ النطق في العربية بساكن ولا ممدود، بل يبدأ بمتحرك" [34] (Khashan, 2003).

The fact that a small number of syllabic structures is allowed by Arabic phonotactics has interesting implications: as the formation of words belonging to the same morphological category is the product of a *quasi* mathematical combination of similar morphemic material, the resulting syllabic structure will tend to follow the same

[34] "In Arabic, speech does not start either with an unvocalized letter (*sakin*) or with a long vowel letter (*mamdood*), but with a vocalized letter (*mutaharrik*)".

patterns. Thus, it seems possible to propose a precise formalism which predicts the syllabic structures for Arabic lexical items.

3.2.2 Stress in MSA

In accordance with the systematization of the phonological feature of lengthening, the position of stress syllables in the lexicon is predictable. It must be determined counting the syllables from the end to the beginning of the word.

(18)

	Arabic	**Transliteration**
word	وَزِيرُ	waziyru
segmentation	وَ - زِي - رُ	wa - ziy - ru
syllable count	3 σ - 2 σ - 1 σ	1 σ - 2 σ - 3 σ

Ryding (2005) provides the following rules for stress position in MSA[35]:

1. Stress is never on last σ—except in pausal-style pronunciation in which, if last σ is super-heavy, stress is on last σ
2. If penultimate σ is strong, stress is on penultimate σ
3. If penultimate is not strong, stress is on antepenultimate σ

Taking into account just the full-style pronunciation, these rules can be reformulated in the following proposition:

if penultimate is not heavy, stress is on antepenultimate;
otherwise, stress is on penultimate

[35] In full-style pronunciation.

3.3 Phonological constraints

It has long been noted that adjacent or proximate homorganic consonants restricted to the Arabic root morpheme are discouraged by Arabic grammar (McCarthy and Prince, 1986; Pierrehumbert, 1993; Padgett, 2001; Frisch et al., 2004; Alderete et al., 2010), i.e. consonants showing similar features tend not to co-occur within the same root. Positional proximity is a variable that gradually encourages the restriction, so the closer the segments the more they are avoided. This property confined to the root consonants was first noted by Greenberg in 1950. McCarthy and Prince described this quality of the root by the Obligatory Contour Principle for Place of Articulation (OCP-Place).

The OCP-Place principle was first proposed by Leben and Goldsmith—in 1973 and 1976 respectively—in studies about tone to explain those kind of restrictions which are confined to a specific autosegmental tier (Watson, 2007). The OCP-Place is a phonological constraint on lexical representations which simply bans adjacent identical elements in a phonological constituent (Berent & Everett, 2001; Pierrehumbert, 1993). Conversely, identical consonants covering the second and third positions of the root are favoured by Arabic phonotactics. To explain this apparent inconsistency, McCarthy and Prince claimed that roots having identical second and third segments are just two-consonant roots in underlying representation and, therefore, the OCP-Place is not applied. Other authors, however, do not support this analysis—for instance Pierrehumbert, 1993[36].

[36] It is not the goal of this thesis to go further on this issue, but just to show the existence of this phonological constraint as supported by several authors.

Many authors, including Greenberg himself, McCarthy (1991) and Stefan et al. (2004) studied the frequency of radical co-occurrences in the Arabic lexicon[37], discovering a non random combination of the root consonants. Their data of the co-occurrence of consonant pairs shows a clear tendency of refusing pairs belonging to the same class, especially when adjacent, but also in nonadjacent positions. Based on their analyses, the cooccurrence restrictions of phonological segments define the following natural classes (Pierrehumbert, 1993; Frisch et al., 2004:185):

Labials	{b, f, m}
Coronal Sonorants	{l, r, n}
Coronal Obstruents	{t, d, s, z, T, D, S, Z, þ, ð, X}
Dorsal Obstruents	{k , q}
Gutturals Approximants	{x, g, H, ç, h, c}

Another interesting case of phonological constraint affecting the Arabic lexicon is related to syllabic structure. As we have seen, consonant clusters at the beginning of a word are not allowed in Arabic. In the same way, syllables without an onset are discouraged due to a phonotactic restriction. There is a bundle of verbs which begin with a consonant cluster. Consequently they add an epenthetic vowel *i* to break the cluster and then insert an epenthetic glottal stop *c* to cover the onset position. Graphically, the segment is represented as *Ai*—recall that the letter alif *A* is used to represent a glottal stop consonant—and does not have morphological status (Brame, 1970; McCarthy, 1993; Beesley, 1998a). The purely phonological nature of this segment is proved by the fact that if the stem is preceded by a vowel in connected speech, it does not require the prosthesis syllabification. For instance, if the form اِنْفَعَلَ *Ainfaçala* is preceded by وَ *wa*, the conjunction equivalent to 'and', the resulting form is وَنْفَعَلَ *wanfaçala*.

[37] Stefan et al., as well as McCarthy, carried out his quantitative analysis of root units combination using the lexicon of roots extracted from the famous Hans Wehr dictionary—they used different versions though. The lexicon of roots used by Stefan et al. contains 2,674 roots.

3.4 Allomorphism[38]

One of the most complicated aspects of Arabic morphology is the analysis of the allomorphic variants. Allomorphism is the situation in which the same morpheme have "different phonological shapes under different circumstances" (Lieber, 2009:22), and all the shapes of that morpheme operate in complementary distribution. In Arabic, allomorphism is mainly caused by the phonological constraints on the semiconsonants *waw* and *ya*. Verbs with roots containing at least one semiconsonant phoneme typically present phonological alterations. In spite of the uniform nature of these phonological alterations, which are susceptible to systematization, verbs suffering these phenomena are considered irregular in traditional Arabic grammar. Nonetheless, it is essential to note that these alterations are by no means arbitrary, but they entail systematizable subregularities.

We understand the alterations in terms of processes which are described by means of morphophonological rules. These rules were "historically phonetically motivated, but affect morphology" (Lieber, 2009:23). As Lieber states, "metaphorically, it is often convenient to think about phonological allomorphy in terms of a single underlying representation that is manipulated by rules under certain conditions. The final result, i.e. what is actually pronounced, is the surface representation" (Lieber, 2009:23).

As we have said, the semiconsonants, *waw* and *ya*, behave in different ways along the inflectional paradigm. The variables that affect their behaviour are:

1. The position of the semiconsonants within the root: the behaviour of the semiconsonant is completely different depending on whether the semiconsonant is in the first position of the root, in the second position or in

[38] We decided to include allomorphism in the section of phonology, instead of locating it in morphology, due to the fact that Arabic allomorphism is highly motivated phonologically.

the third. Apart from this, semiconsonants in quadriliteral roots tend to be more regular than those of triliteral roots.

2. The pattern of the verb. This has to do with the syllabic structure each pattern presents.
3. The vocalic melody surrounding the semiconsonants, which is related to the vocalism morpheme of each pattern.
4. The presence of more than one semiconsonant in the verbal root: roots with two semiconsonants tend to keep one of them as a *regular behaving* phoneme.
5. There is a general constraint rule in Arabic phonotactics which states that the combination *wi* is systematically prohibited, and thus commonly avoided.

Verbs containing at least one semiconsonant in its root are known as *weak verbs* الفعل المعتلّ. If the first letter of the root is weak, the verb is called *assimilated* الفعل المثال; if the second, the verb is called *hollow* الفعل الأجوف, if the third, it is known as *defective* الفعل الناقص.

The so-called geminated or doubled verbal roots الفعل المضعّف are another major cause of allomorphism. These verbal roots are characterized by containing identical segments in their second and third radical positions.

There is another interesting case of phonological constraint which is restricted to the derivational infix -t-, present in verbs of pattern VIII. This infix may assimilate with the first consonant of the verbal root. The assimilation is systematic and depends on the combination of segments. For instance, the -t- followed by an emphatic consonant will assimilate the emphatic feature, and becomes -T-. A complete formalization of the assimilation rules is shown in results-2.2.5 , as part of the generation model.

Finally, we must also incorporate the orthographic component. There are some graphic alterations with no linguistic content which, despite being exclusively orthographic, need to be treated by means of contextual rules. In this sense, we can

refer to them as *orthographic allomorphism*. The set of orthographic allomorphs is virtually always caused by the presence of the letter hamza in the root. These types of roots are called *hamzated roots*. We saw that there are six different graphic symbols to represent this letter and that their shape is determined by be surrounding context.

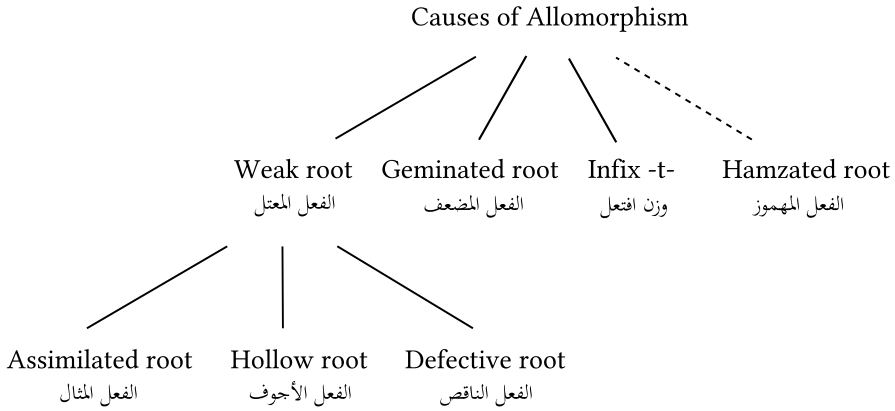


FIGURE 5 Causes of allomorphism

The line of the hamzated root is dotted to highlight the fact that it is not phonological allomorphism, but orthographic.

In the following lines we present some examples of the phonological segments described above which cause allomorphism. Each of the examples from (19) to (24) illustrates one of the causes of allomorphism described above. Examples (25) and (26) present two cases in which more than one type of allomorphism is registered in a form.

cause of allomorphism	root	underlying form	surface form	gloss
(19) assimilated root	وعد wçd	تَوْعِدْنَ taw·çid·na	تَعِدْنَ taçid·na	they (fem.) promise
(20) hollow root	ضيف Dyf	يُضِيفُ yuDiyifu	يُضِيفُ yuDiyfu	he adds
(21) defective root	عطو çTw	يُعْطُوونَ yuç·Tiwuwna	يُعْطُونَ yuç·Tuwna	they (masc.) give
(22) geminated root	حبب Hbb	نُحْبِبُ nuH·bibu	نُحِبُّ nuHib~u	we love
(23) infix -t- pattern VIII	وصل wSl	اوْتَصَلَ Aiw·taSala	اتَّصَلَ Ait~aSala	he contacted
(24) hamzated root	ءكد ckd	أَءَكِّدُ Áucak~idu	أَؤَكِّدُ Áuúak~idu	I confirm
(25) geminated root and infix -t- pattern VIII	ضرر Drr	أَضْتُرِرُ AuD·turir·	أَضْطُرُّ AuD·Tur~a	he was obliged
(26) hollow and hamzated root	جوء jwc	جَوَءْتُ jawac·tu	جِئْتُ jiý·tu	I came

TABLE 6 Examples of causes of allomorphism

4 Morphology

4.1 Basic concepts

Morphology is the study of the internal structure of words and the systematic covariation between that structure and the meaning of the *word*[39] (O'Grady et al., 1996; Al-Sughaiyer and Al-Kharashi, 2004; Jurafsky & Martin, 2009; Haspelmath & Sims, 2010). Words are constituted by morphemes, which can be defined as "the smallest meaningful constituents of words" (Haspelmath & Sims, 2010), so morphemes are seen as atomic elements, i.e., they cannot be split into smaller units (Spencer, 1991).

The main goals of morphological analysis are, on the one hand, to provide an elegant and cognitively realistic description of word structure and language grammar as a whole. On the other, it tries to find a grammatical theory common to all languages which allows us to understand language as a human ability, as languages are instances of this ability. At the same time, by describing all languages under the same descriptive architecture, it would allow us to freely compare the linguistic structures of different languages amongst themselves (Haspelmath & Sims, 2010).

Morphology also studies how morphemes combine with each other inside words. This part of morphology is named *morphotactics* (Attia et al., 2011; Haspelmath & Sims, 2010). Morphotactics, thus, is concerned with the form of morphemes and presents

[39] There is no agreement in the linguistic community on how to define the concept of word. We are not going to try to provide a technical definition for it, as we do not find it necessary for the present work. For the sake of convenience, we are simply going to adopt a non-linguistic definition of word for the following sections, which is more suitable for computational purposes: we will think of a word as a sequence of letters with no spaces between them.

two strategies: concatenative and non-concatenative—also known as templatic. Languages make use of one or two of the strategies depending on the configuration of its morphological system.

Concatenative morphemes are those which can be segmented as a unit, i.e., they have the form of an uninterrupted string.

Non-concatenative morphemes are interleaved elements within a word—they do not form a compact unit, but a discontinuous string—whose 'internal blanks' are filled out with other morphemes, so that the lexical material is formed by intertwined morphemes.

A *word-form* is a word in a concrete sense, whereas a *lexeme* is a word in an abstract sense. (Haspelmath & Sims, 2010). "*Lexemes* can be though of as families of words that differ only in their grammatical endings or grammatical forms" (Lieber, 2009). Thus, in a concrete sense, we can think of lexemes as a set of word-forms corresponding to the same word, i.e., the referential meanings of all the word-forms in the set are the same. Typically, one of the word-forms comprising a lexeme is considered the unmarked form and thus used as a dictionary entry. It may happen that a lexeme only exhibits one form, in which case the lexeme representation corresponds to the word-form. Lexemes are also-called *lemmas*. The difference between both terms simply lies in the field of studies in which they are used: the term *lexeme* is used in theoretical studies, whereas *lemmas* is used in practical works about lexicography and in computational linguistics.

"The set of word-forms that belongs to a lexeme is often called a *paradigm*" (Haspelmath & Sims, 2010). The paradigm of a lexeme exhibits the different endings that the set of word-forms from that lexeme may have. The paradigm of a noun or adjective is traditionally called *declension* and that of a verb *conjugation* (Lieber, 2009).

The relation among the word-forms of a lexeme is called *inflection*. Inflectional word formation typically expresses grammatical distinctions such as number, tense, person and case, among others. (Lieber, 2009). It is generally said that it does not result in the creation of new lexemes, "but merely changes the grammatical form of lexemes to fit into different grammatical contexts" (Lieber, 2009; Al-Sughaiyer and Al-Kharashi, 2004).

The morphemes attached—in the case of concatenative morphologies—to word-forms to create lexemes are called *affixes*. Affixes can be attached to a word-form at the beginning (*prefixes*), at the end (*suffixes*), around the word-form (*circumfixes*), in the middle (*infixes*), or in various parts of the word-form structure, as a discontinuous morpheme (*transfixes*). If we remove the affixal material from the word-form, the remaining structure is called *stem* or *base* (Lieber, 2009).

A set of lexemes, in turn, may be related structurally, by means of other kinds of affixal morphemes bearing semantico-referential meaning. The relation among a set of associated lexemes is called *derivation*. Again, from a concatenative perspective, derivational affixes are classified as prefixes, suffixes, circumfixes, infixes and transfixes.

Inflection is, therefore, the relationship among the word-forms of a lexeme, whereas derivation is the relationship among lexemes of a word family (Lieber, 2009). Inflection and derivation are thus related to the semantic content of morphemes. We have said that inflection typically does not change the part of speech of the word it applies, whereas derivation often changes the word class. Derivation tends to be applied closer to the stem, and inflection is usually more external. Another difference between the two is that inflection has a more abstract meaning and is relevant to syntax, it is typically pure grammatical material which represents the morphosyntactic relations of the constituents of the language. On the contrary, derivation is more concrete and not relevant to syntax (Haspelmath & Sims, 2010).

Although the distinction between inflection and derivation is mostly clear, there are cases in which this distinction is blurred.

There are some words which do not have independent phonological status and are always bound to another word. However, the relationship between both words is purely syntactic. Orthography treats these forms writing them together—as a reflection of their lack of phonological stress—and therefore these bound morphemes, known as *clitics*, will have the same representation as inflectional and derivational morphemes. Clitics are always concatenative, and may be attached at the beginning (*preclitics*) or at the end (*posclitics*) of words.

As we have outlined above, the core task of morphological analysis of any language is to separate and identify the component morphemes of a word (Beesley, 2001) and the relations established among them. Morphotactics studies how these morphemes combined and what form they have. Then, we can classify these morphemes according to their semantic content in *derivational* or *inflectional*.

4.2 Arabic morphotactics

Arabic morphology uses both concatenative and non-concatenative strategies. Concatenative morphemes are discrete segments which are simply attached to the stem regardless of the position. For instance, the corresponding feminine gender suffix ـة- –ä is attached at the end of Arabic stems:

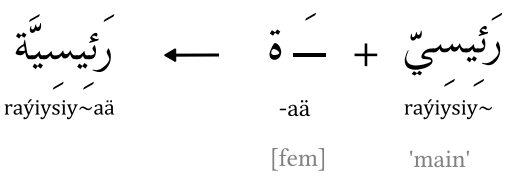


FIGURE 6 Example of concatenative process

Arabic, as is typical of Semitic languages, also makes extensive use of non-concatenative procedures to build stems. Many Arabic stems are composed of a Semitic root and a pattern. The *root*[40] جذر √jðr is a decomposable morpheme that provides the basic meaning of a word. It therefore contains lexical meaning and usually expresses the notion of an act, so it conveys the idea of 'doing something', but also state and process. Generally, verbal roots consists of 3 or 4 ordered consonants in non-linear position within the word. The consonant components of the root are traditionally known as *radicals*. Roots are said to be inserted in patterns to create stems. One group of patterns accepts three radicals and another four radicals. Three-consonantal roots are known as *triliteral* in the field, e.g. بيع √*byç* 'selling', whereas four-consonant ones are called *quadriliteral* roots, e.g. ترجم √trjm 'translating' (Cowan, 1958; Versteegh, 1997; 2001; Shimron, 2003; Beesley, 1998a and 2001; Pierrehumbert, 1993; Habash, 2010; Cavalli-Sforza et al., 2000).

The *pattern* is a syllabic structure which contains vowels—and sometimes consonants too—in which the radicals of the root are inserted and occupy specified places. (Attia et al., 2011; Beesley, 2001). This type of Arabic—and Semitic—morphology, which consists of the interdigitation of roots and patterns to form stems, is conventionally called *root-and-pattern* morphology (McCarthy, 1981; Cowan, 1958; Erwin, 1963; Abu-Chacra, 2007; Beesley, 1998b and 2001).

In the literature *pattern* is also termed *template, binyan*—taken from Hebrew—and its plural form *binyanim*; the Arabic equivalent *wazn* وزن, plural *awzan* أوزان, meaning 'measures'; *measure* itself, and *derived forms*. In the field of computational linguistics, all those terms tend to be ambiguously used to represent the pattern with or without the vocalic material, i.e. specifying the vowel qualities directly in the pattern or not. In order to use accurate terminology, from now on we will use the term *pattern* to

[40] From a concatenative-like perspective, roots are considered transfixes.

refer to the derivational morpheme including the vocalic material, and the term *template* without the vocalic material.

The following figure shows an abstract representation of the root and pattern interdigitation.

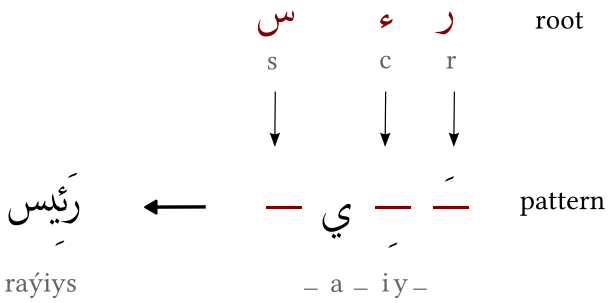


FIGURE 7 Example of non-concatenative process

Broadly speaking, the affixal material included in the patterns corresponding to the vowels bears inflectional meaning, whereas the consonants add extensions to the basic meaning expressed by this root.

In this sense, the non-linear vowel string is also said to form a special morpheme. As McCarthy stated, "consonantal roots and vocalic melodies in Arabic, although they contain bundles of the same distinctive features, can nevertheless be represented on separate autosegmental tiers." (McCarthy, 1981)

Therefore, patterns may be split into two separate items: a CV-scheme or template and a vowel melody or vocalism (Soudi & Eisele, 2004). The Cv-scheme may or may not contain consonantal affixes of derivational nature. The following figure shows this abstract representation of Arabic stems, following the lines of templatic morphology—in this case, the pattern does not include consonantal affixes:

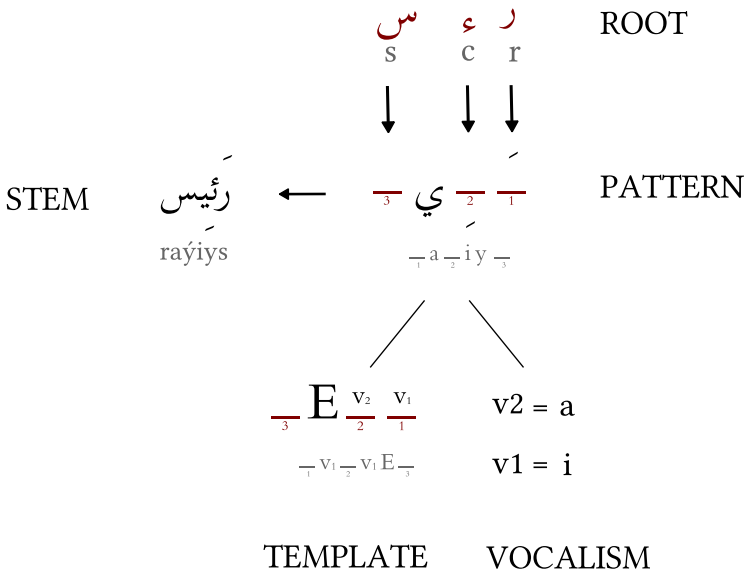


FIGURE 8 Example of complete decomposition of Arabic stem
The symbol 'E' is used to indicate vowel lengthening

In relation to the association of the semantic content of the different morphemes and their formal properties, derivational morphology is mainly marked by non-concatenative schemes, whereas inflectional morphology tends to be concatenative. However, there are various cases in which the opposite occurs.

The most productive example of a derivational morpheme following a concatenative strategy is the so-called *nisba* relation, which makes relational adjectives by the addition of the suffix -iy~ 'ـِيّ-'.

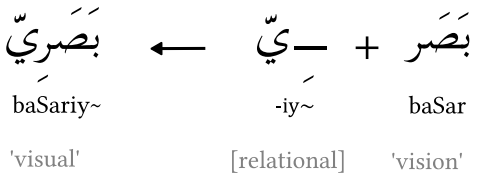


FIGURE 9 Example of derivational morpheme formed by concatenative process

In contrast, perhaps the most productive instance of inflectional morphology formally represented by templatic procedures is the so-called *broken plural*. Arabic has a fairly large inventory of patterns for forming plural forms from nouns and adjectives (Ratcliffe, 1998; Souag, 2002). Another interesting case is the feminine—not very productive—pattern { _ a _ _ a Y }. For instance:

(27)

	masculine	**feminine**[41]
Arabic	كَسْلَان	كَسْلَى
transliteration	kas·laAn	kas·laY
gloss	lazy (mas.)	lazy (fem.)

Arabic verbal morphology shows an unequivocal relation between form and function, making the analysis of verbs quite straightforward in this sense. Contrary to verbal morphology, nominal morphology presents several inconsistencies in this relation, which is indeed very natural; for instance, a similar situation has been noted in the Spanish language (Moreno Sandoval & Goñi Menoyo, 2002). A specific function tends to be prototypically bound with a concrete form, yet it may adopt other forms in marked contexts. Similarly, one same form may be used to express different relations in different situations. This type of affixes are known as multifunctional ambiguous affixes. The matrix of formal and functional relationships established amongst

[41] The form كَسْلَانَة *kas·laAnaö* is also accepted as feminine.

morphosyntactic elements is in fact very complex—to cite an instance of works on this topic, see Ferrando (2006).

For instance, the inflectional feature of feminine gender is prototypically assigned to the suffix form ‘ـَة–’–aä, as in:

(28) Form inflectionally marked for feminine by suffix:

Arabic	وَالِد + ـَة = وَالِدَة
Transliteration	waAlid + aä = waAlidaä
Gloss	*mother*

However, lexical forms marked for feminine gender may not present the suffix –aö.

(29) Form lexically marked for feminine:

Arabic	أُمّ
Transliteration	Áum~
Gloss	*mother*

(30) Form inflectionally marked for feminine by pattern:

Arabic	خَضْرَاء	وزن: فَعْلَاء
Transliteration	xaD·raAc	pattern: { _ a _ _ a A c }
Gloss	*green (fem.)*	

On the other hand, the same suffix–aö may be used in forms not marked for feminine, either because the same suffix is used for another function or because the word is lexically masculine and thus intrinsically cannot present the feature feminine.

(31) Suffix –aö marking singular number

Arabic	حَجَر + َة = حَجَرَة
Transliteration	Hajar + aö = Hajaraö
Gloss	*a rock*

(32) Suffix –aö bearing no morphological meaning, because the word is lexically masculine

Arabic	خَلِيف + َة = خَلِيفَة
Transliteration	xaliyf + aö = xaliyfaö
Gloss	*successor (man), caliph*

This is the general scheme of Arabic morphology:

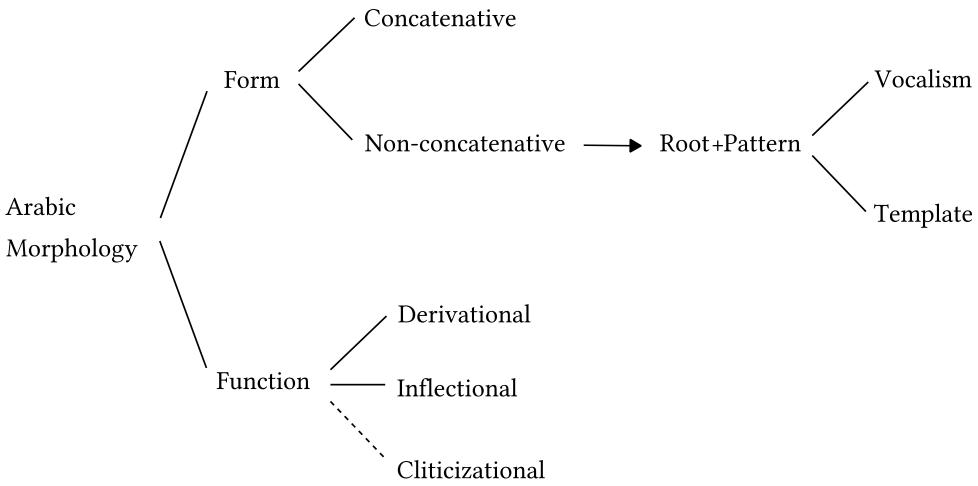


FIGURE 10 General framework of Arabic morphology

(33)

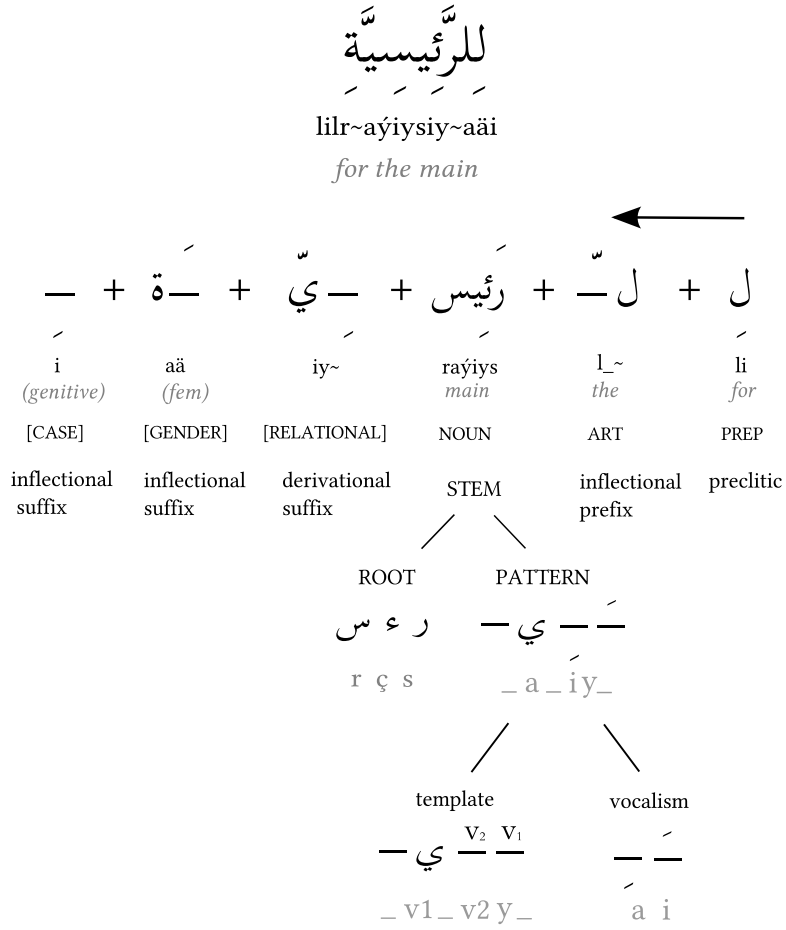


FIGURE 11 Example of complete analysis of wordform

4.3 Arabic prosodic morphology

Following the deep analysis scheme, McCarthy and Prince (1990), outlined a theoretical description of Arabic morphology, called autosegmental morphology. Autosegmental morphology is an adaptation of the theory of autosegmental phonology developed by Goldsmith in 1976, initially to deal with phonological processes such as tone or vowel harmony (Goldsmith, 1979). Both theories propose an analysis with different levels of lexical representation, i.e. a non-linear representation of words. It attempts to provide a simple and efficient way to handle non-concatenative linguistic processes which, though in reality intermingled, seem to operate at different linguistic layers. The theory thus treats the different processes as independent layers—frequently known as tiers—which interact with each other in a methodical way, by means of well-formed associations.

In the words of Ratcliffe, "the autosegmental analysis decomposes a surface phonological string into two or more autonomous strings which are superimposed on each other. Each of these autonomous strings—termed autosegments—will specify a unique set of feature values and will be represented on its own tier" (Ratcliffe, 1998:25). "(...) In McCarthy's system morphemes are not treated as sequences of phonemic segments separated by boundaries. He proposed instead "that the string of segments is uninterrupted, but the morphological analysis is given by another, simultaneous level of representation" (Ratcliffe, 1998:26).

In this model, Arabic stems are represented by the three types of morphemes we have mentioned in the previous section: consonantal roots, vocalism morphemes and a template consisting of a CV-skeleton. Additionally, some stems include affix morphemes. "Each morpheme sits on its own autonomous tier in the autosegmental model; the morphemes are coordinated with association lines according to the

principles of autosegmental phonology" (Kiraz, 1994b). Under a root-and-prosody analysis, templates are believed to be given by prosody, i.e. they "are not axiomatic morphological entities, but rather should be derived from the interaction of prosodic well-formedness constraints with segmental faithfulness considerations" (Tucker, 2009).

The main advantage of Prosodic Morphology (McCarthy 1981; 1993; 1990; McCarthy and Prince, 1990) is that it has proved to be very adequate for describing nonconcatenative phenomena such as reduplication, infixation and templatic morphology (McCarthy and Prince, 1986; Kiraz, 1996). Prosodic Hierarchy defines the actual units of prosody out of which templates must be constructed (McCarthy and Prince, 1986). Watson (2007:56) describes the principles of this hierarchical model in the following manner:

"The three fundamental theses of Prosodic Morphology:

1. Prosodic Morphology Hypothesis. Templates are defined in terms of the authentic units of prosody: mora (μ), syllable(σ), foot (F), phonological word (ω) and so on.

2. Template Satisfactory Condition. Satisfaction of templatic constraints is obligatory and is determined by the principles of prosody, both universal and language-specific.

3. Prosodic Circumscription of Domains. The domain to which morphological operations apply may be circumscribed by prosodic criteria as well as by the more familiar morphological ones" (Watson, 2007:129). "Prosodic circumscription can be either positive or negative, depending on how the prosodically-delimited substring is targeted. In negative prosodic circumscription, some prosodic constituent at the edge of a form is

> disregarded and the morphological operation applies to the remainder" (McCarthy, 1993).

In Prosodic Morphology, the analysis of syllabic structure is based on the mora, i.e., the unit of syllabic weight. We have already noted that the most common types of syllables in Arabic are a monomoraic (light) syllable CV and two kinds of bimoraic (heavy) syllables CVV and CVC. Prosodic morphology establishes that "the minimal syllable is monomoraic with an obligatory onset" (Watson 2007:56).

4.4 The system of Arabic verbal morphology

4.4.1 General overview

Traditionally, Arabic verbs are divided into verbs formed from triliteral roots and verbs formed from quadriliteral roots. At the same time, each group is further divided into *simple patterns* الأوزان المجردة, which contain no derivational material, but just the root, and *derived patterns* الأوزان المزيدة, in which the root is *augmented* with additional derivational material. Triliteral root verbs have a total of 16 hypothetical patterns, i.e., 16 derivational classes: one simple pattern and 15 derived patterns. In turn, quadriliteral root verbs have theoretically one simple verb pattern and 3 derived pattern. There is no a single root, either triliteral or quadriliteral, which have verbs formed from all the patterns. All patterns show different vowel melodies in their stem (Wright, 2007).

Simple verbs, however, may adopt more than one different stem. This is because there are variations in the vocalic qualities of the simple patterns depending on the verb—in

the rest of the patterns this does not happen. The possible combinations of perfective and imperfective patterns for each verb, considering vowels, provide 6 possible verbal patterns (El-Dahdah, 1991) [42].

triliteral root		quadriliteral root	
simple patterns	derived patterns	simple patterns	derived patterns
6	14	1	3

TABLE 7 Number of verbal patterns

Inflectional tense/aspect and voice are marked by changes in the stem vowelling. The former is also marked by a specialization in the inflectional paradigm: perfective الماضي and imperfective المضارع conjugations have different sets of affixes. In addition, the imperative paradigm, though related to imperfective, present a different set of inflectional affixes and a different stem shape. Consequently, each verb pattern distinguishes three stems: perfective (*p-stem*), imperfective (*i-stem*) and imperative stem (*m-stem*).

The traditional way of specifying each pattern shape is using the Arabic root فعل *√fçl* meaning *doing*, for triliteral roots, and the convention فعلل *√fçll*, for quadriliterals. In the case of quadriliterals, since the third and fourth radicals are the same, this can lead to much ambiguity, therore in this work we have decided to use the convention فعلب *√fçlb* for quadriliteral roots.

model for triliteral root	فعل	√fçl
model for quadriliteral root	فعلب	√fçlb

Conventionally, Arabic verb lemmas, i.e. the citation form of a verb, correspond to the third person masculine singular of the perfective conjugation; e.g. for simple verbs

[42] The division of the simple pattern in six types is not common.

the citation form is فَعَلَ *façala* meaning '(he) did'. This is because this form is constructed by simply adding a suffix -a to the perfective stem. As the vowel *a*, a diacritic in Arabic, is not written, the form orthographically seems to coincide with the perfective stem of the verb. The p-stem is traditionally considered basic in contrast with the i-stem.

Below is a list of the stems formed from all the theoretical patterns of triliteral and quadriliteral roots, including perfective, imperfective and imperative verbal stems in active voice; the passive differs in the vocalization. Following the Arabic western linguistic tradition, we used Roman numerals to refer to patterns—I is used for simple verbs; II-XV are used for derived patterns. However, we include some additions to make it unequivocal. A letter Q is added at the beginning of the quadriliteral root patterns, so that they are not confused with triliteral root patterns—this is also used in Danks (2011). In addition, the six types of patterns I of the triliteral root have two vowels, the first indicates the thematic vowel of the perfective, the second indicates the thematic vowel of the imperfective—both correspond to the second vowel of the stem. Here, we do not represent the sukun grapheme—recall that it marks the absence of a vowel after a consonant—as it does not have any linguistic content. Still, in a computational model we will have to deal with, but in this part we believe it unnecessary and even counterproductive. Additionally, we are not going to represent the segment ـْـِ 'Ai', present in some verbal forms—from VII to X, QIII and QIV.

Pattern		p-stem		i-stem		m-stem	
		Arabic	trans.	Arabic	trans.	Arabic	trans.
triliteral	Iau	فَعَل	façal	ـَفعُل	afçul	فعُل	fçul
	Iai	فَعَل	façal	ـَفعِل	afçil	فعِل	fçil
	Iaa	فَعَل	façal	ـَفعَل	afçal	فعَل	fçal
	Iuu	فَعُل	façul	ـَفعُل	afçul	فعُل	fçul
	Iia	فَعِل	façil	ـَفعَل	afçal	فعَل	fçal
	Iii	فَعِل	façil	ـَفعِل	afçil	فعِل	fçil
	II	فَعَّل	faç~al	ـُفَعِّل	ufaç~il	فَعِّل	faç~il
	III	فَاعَل	faAçal	ـُفَاعِل	ufaAçil	فَاعِل	faAçil
	IV	أَفعَل	çafçal	ـُفعِل	ufçil	فعِل	fçil
	V	تَفَعَّل	tafaç~al	ـَتَفَعَّل	atafaç~al	تَفَعَّل	tafaç~al
	VI	تَفَاعَل	tafaAçal	ـَتَفَاعَل	atafaAçal	تَفَاعَل	tafaAçal
	VII	نفَعَل	nfaçal	ـَنفَعِل	anfaçil	نفَعِل	nfaçil
	VIII	فتَعَل	ftaçal	ـَفتَعِل	aftaçil	فتَعِل	ftaçil
	IX	فعَالّ	fçalal	ـَفعَلِل	afçalil	فعَلِل	fçalil
	X	ستَفعَل	stafçal	ـَستَفعِل	astafçil	ستَفعِل	stafçil
	XI	فعَالَل	fçaAlal	ـَفعَالِل	afçaAlil	فعَالِل	fçaAlil
	XII	فعَوعَل	fçawçal	ـَفعَوعِل	afçawçil	فعَوعِل	fçawçil
	XIII	فعَوَّل	fçaw~al	ـَفعَوِّل	afçaw~il	فعَوِّل	fçaw~il
	XIV	فعَنلَل	fçanlal	ـَفعَنلِل	afçanlil	فعَنلِل	fçanlil
	XV	فعَنلَى	fçanlaA	ـَفعَنلِي	afçanliy	فعَنلِي	fçanliy
quadriliteral	I	فَعلَب	façlab	ـُفَعلِب	ufaçlib	فَعلِب	façlib
	II	تَفَعلَب	tafaçlab	ـَتَفَعلَل	atafaçlab	تَفَعلَل	tafaçlab
	QIII[43]	فعَنلَب	fçanlab	ـَفعَنلِب	afçanlib	فعَنلِب	fçanlib
	QIV	فعَلبَب	fçalbab	ـَفعَلبِب	afçalbib	فعَلبِب	fçalbib

TABLE 8 List of all verbal patterns and their stems

[43] Forms QIII and QIV are sometimes exchanged in literature.

Patterns IX, XI and QIV present an interesting peculiarity: they have two different stem [44] forms in each stem of their inflectional paradigm. For instance, the imperfective form in 3rd person masculine singular of pattern IX يَفعَلُّ *yafçal~u* and 3rd person feminine plural يَفعَلِلنَ *yafçalilna* present different stems; the former has a stem *afçal~* and the latter *afçalil*. Various authors consider that the first stem is built from the second one by a process of consonant spreading of the final radical (McCarthy, 1981; Kiraz, 1994a; Bird and Blackburn, 1991, Beesley, 1998a). Thus, the underlying form of pattern IX i-stem would be /afçalil/. Here, we have represented the stems of pattern IX, XI and QIV in accordance with this convention.

Patterns XI, XII, XIII, XIV, XV, QIII and QIV are only present in Classical Arabic and, even within this variety, they are rarely found in the lexicon[45].

In relation to meaning, the derived patterns are said to present some regular semantic connotations, but they must be understood simply as tendencies. These semantic tendencies imply, at least in some cases, modifications to the basic meaning of the root. There are many studies trying to establish the exact meanings expressed by the different derivational material, according to each of the traditional patterns. They have been generally described as follows (Ryding, 2005; Wright, 2007; Danks, 2011):

I	Its meaning is usually the most approximate to the basic meaning of the root
II	Causative of transitive pattern I
	Transitive of intransitive pattern I
	Intensive or iterative

[44] Recall that the root is a lexical morpheme containing the basic meaning of a word and which consists of 3 or 4 ordered consonants in non-linear position within the root. On the other hand, the stem is the result of removing the affixal material to the word-form. This remaining strucuture constitutes the stem.

[45] In Appendix B, we provide the frequency of each pattern according to roots.

	Denominative
	Estimative
III	Associative
	Reciprocal action
	Iterative
	Attempted action
IV	Causative or double transitive of transitive pattern I
	Transitive of intransitive pattern I
	May have meanings similar to pattern II
	Declarative/estimative
	Denominative of place names
V	Reflexive of pattern II, often referred to as mediopassive
	The derivational prefix -ta has recently been said to be an unaccusative construction
	Resultative of pattern II
	Gradual progress in an activity or state
VI	Reciprocal of pattern III
	Reflexive of pattern III
	Denotes a pretence
VII	Reflexive pattern I
	Passive of pattern I
	Resultative
	Mediopassive
	Ergative and unaccusative constructions in Arabic
VIII	Reflexive or mediopassive of pattern I
	Resultative of the action of a pattern I
	(Contrary to Form VII) it may take a direct object
	(Occasionally there is no difference in meaning between patterns I and VIII)
IX	Acquisition of a colour or a physical trait
	Usually deadjectival verbs from the nominal pattern أَفْعَل *Aafçal*

X	Requestative or estimative
	Reflexive of pattern IV
XI	Acquisition or existence of a color or physical trait. Wright thinks that forms IX and XI are indistinguishable in sense
XII	Colour or physical quality
XIII	Usually denotes colour or quality but may also denote an action
XIV	colour or physical quality
XV	-
I	It may be transitive or intransitive
II	Reflexive
	Resultative
	Passive of Form I
	Denominative—very productive for new loanwords
III	Corresponds in meaning to pattern VII of the triliteral roots
IV	Corresponds in meaning to pattern IX of the triliteral roots

Danks (2011) studied the meaning of each pattern by analyzing the co-occurrences of pattern pairs using the lexicon of verbs extracted from the Hans Wehr dictionary. He started from the premise that certain verbal patterns are morphologically related—as traditionally stated—and thus validated this hypotheses using quantitative data: "since phonological constraints are minimal, it is likely that the frequency with which a pattern occurs depends on how semantically or syntactically *useful* it is. Nevertheless, if the patterns are all independent of one another, we would expect to see them distributed randomly amongst the roots" (Danks, 2011:31). He did not study all the patterns, but he focused his research on vowel lengthening patterns. One of the properties he analyzed is transitivity, which he defined as a continuum, not as a binary opposition as is often understood. Arabic accepts direct objects through prepositions, double transitive verbs and ambitransitivity—transitive verbs used as intransitive, i.e. the direct object is optional. For example, a double transitive verbal pattern deriving another pattern that just allows one direct object may be seen as a

reduction in the degree of transitivity, though both patterns are transitive in a dichotomic representation. If the detransitivising process is morphological and not lexical, this may provide us some hints about the meaning of both patterns.

In analyzing his data, Danks encountered a clear correlation between II-V, III-VI and QI-QII forms. The distinction of all these pairs is made by the presence of the prefix ta-; forms V, VI and QII are derived from II, III and QI by means of a prefix –ta which involves a detransitivising process. The highest degree of correlation was found in the III-VI pair. Further, he states that in most of the cases—around 70%—pattern III expresses mutuality of action, while its related pattern VI expresses reciprocity. Interestingly, he found no process of transitivity in pattern II from base meaning.

Nevertheless, it is important to take into account that the meaning of a verb is suggested by composition of the root meaning plus the pattern meaning, but it does not have compositional meaning *per se*. In this respect, McCarthy claimed "(...) the root supplies the basic meaning and the binyan [pattern] (except for the first binyan) supplies some modification of this meaning or of the verbal diathesis. The meaning of any verb is not a composition of the meaning of root and binyan, but there is a reasonable amount of predictability" (McCarthy, 1981). For instance, even though several patterns tend to include transitive or causative verbs, it has been suggested that these semantic features are a lexical property of roots (Hallman, 2006). Dichy goes further and deems the patterns to be insufficient basis for the relations established between the stem and the word-forms: "This view of the ROOT & PATTERN structure can be described as mythical, and operates as a heavy epistemological impediment, reducing relations between grammar and lexicon within the Arabic word-form almost to nothing, since the PATTERN is in itself a grammatical morpheme." (Dichy, 2000:57)

Arabic inflectional system distinguishes tense/aspect (perfective and imperfective), voice (active and passive), gender (masculine and feminine), number (singular, dual

and plural), person (first, second and third) and mood (indicative, subjunctive, jussive and imperative) [46]. Mood is only marked in imperfective, never in perfective.

In the following table a complete conjugational paradigm is shown. Inflectional affixes are separated by a hyphen in the transliteration.

Inflec. Info	Perfective	Imperfective			Imperative
		Nominative	Subjunctive	Jussive	
3SM	فَعَلَ façal-a	يَفعُلُ y-afçul-u	يَفعُلَ y-afçul-a	يَفعُل y-afçul	
3SF	فَعَلَتْ façal-at	تَفعُلُ t-afçul-u	تَفعُلَ t-afçul-a	تَفعُل t-afçul	
3DM	فَعَلَا façal-aA	يَفعُلَانِ y-afçul-aAni	يَفعُلَا y-afçul-aA	يَفعُلَا y-afçul-aA	
3DF	فَعَلَتَا façal-ataA	تَفعُلَانِ t-afçul-aAni	تَفعُلَا t-afçul-aAni	تَفعُلَا y-afçul-aA	
3PM	فَعَلُوا façal-uwA	يَفعُلُونَ y-afçul-uwna	يَفعُلُوا y-afçul-uw(A)	يَفعُلُوا y-afçul-uw(A)	
3PF	فَعَلْنَ façal-na	يَفعُلْنَ y-afçul-na	يَفعُلْنَ y-afçul-na	يَفعُلْنَ y-afçul-na	
2SM	فَعَلتَ façal-ta	تَفعُلُ t-afçul-u	تَفعُلَ t-afçu-la	تَفعُل t-afçul	فعُل fçul
2SF	فَعَلتِ façal-ti	تَفعُلِينَ t-afçul-iyna	تَفعُلِي t-afçul-iy	تَفعُلِي t-afçul-iy	فعُلِي fçul-iy
2DN	فَعَلتُمَا façal-tumaA	تَفعُلَانِ t-afçul-aAni	تَفعُلَا t-afçul-aA	تَفعُلَا t-afçul-aA	فعُلَا fçul-aA
2PM	فَعَلتُم façal-tum	تَفعُلُونَ t-afçul-uwna	تَفعُلُوا t-afçul-uwA	تَفعُلُوا t-afçul-uwA	فعُلُوا fçul-uwA
2PF	فَعَلتُنَّ façal-tun~a	تَفعُلْنَ t-afçul˜na	تَفعُلْنَ t-afçul-na	تَفعُلْنَ t-afçul-na	فعُلْنَ fçuln-a

[46] In Classical Arabic, there is another mood which is rarely used even in this variety, the so-called energetic. We are not going to consider it in our description.

1SN	فَعَلتُ façal-tu	أَفعَلُ Á -afçul-u	أَفعَلَ Á-afçul-a	أَفعَل Á -afçul
1PN	فَعَلنَا façal-naA	نَفعَلُ n-afçul-u	نَفعَلَ n-afçul-a	نَفعَل n-afçul

TABLE 9 Complete table of verbal conjugation

The table shows the Arabic inflectional system for verbs. The grammatical categories involved in conjugation are: aspect/tense, voice, mood, person, number and gender. For the tags used to express person, number and gender:

First position: 1=first person; 2=second person; 3=third person;

Second position: S=singular; D=dual; P=plural;

Third position: M=masculine; F=feminine; N=non-marked for gender

4.4.2 Derivational morphemes

As we outlined previously, Arabic has been traditionally described as a paradigmatic example of root-and-pattern morphology, a variety of non-concatenative morphology in which the constituents of the word cannot be segmented into compact units. Morphemes are not attached to words in the form of affixes, but as discontinuous elements.

The basic constituent of a Semitic word is the root, which we defined as a set of 3 or 4 ordered but non-concatenative consonants that contains the basic meaning of the word. However, as Badawi emphasises, it is common for roots to convey quite different meanings: "although many roots embody a single semantic notion, or at least a coherent group of related notions, it is not unusual for a root to contain not only different but even contradictory meanings, though these are mostly rendered unambiguous by context" (Badawi et al., 2004:26). Patterns are *molds* of discontinuous sets of consonants and vowels arranged in a syllabic structure. The radicals of the root are inserted in those patterns. The consonantal material included in the

patterns—together with vocalic lengthening processes—constitutes derivational affixes. With the combination of these two elements—the root and the pattern—Arabic stems are created.

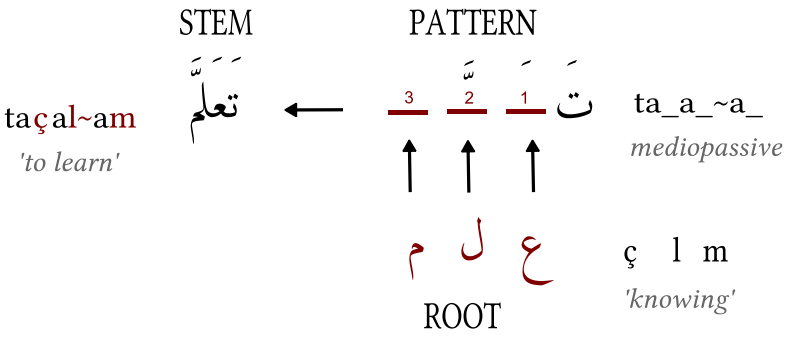


FIGURE 12 Root-and-pattern morphology

We have seen that some theoretical proposals attempt to analyze the stem at a deeper level, and thus split the pattern in three items: (1) a syllabic 'CV' structure with the content of the consonants of the pattern, (2) a root string and (3) a vocalic string or vocalism. Additionally, some consonantal affixes can be inserted into the template. The syllabic structure, or template, refers to the structure of stems according to types of syllables that have been measured in moraic units. Each type of verb has different affixal consonants inserted in the template. These constitute the derivational material of each derived pattern. Vowels have to be inserted into specific slots of the syllabic structure. Their quality depends on lexical and grammatical characteristics.

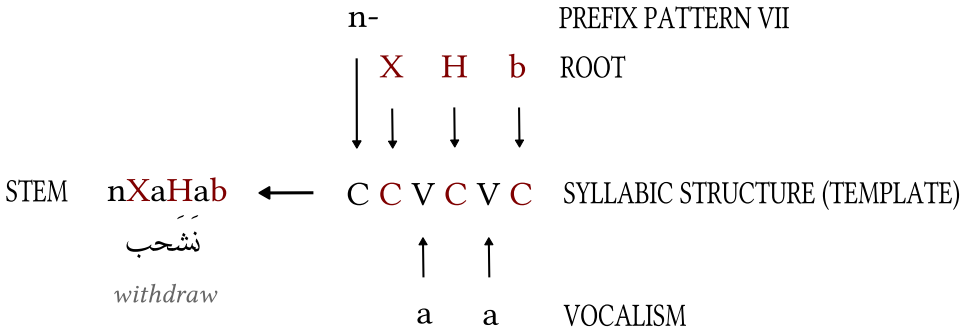


FIGURE 13 Decomposition of stem

This three level analysis has been supported by many authors oftentimes, to the detriment of the classical root-and-pattern analysis. As Ratcliffe states "it has become clear in the last half century, or so, that the root and pattern analysis alone is insufficient in terms of modern notions of descriptive adequacy". (Ratcliffe, 1998:23)

4.4.2.1 The root morpheme

The concept of Semitic root must be understood and used as a morphological tool under a synchronic perspective, never as a means of diachronic analysis. Yet, it is often the case that works on the historical evolution of Arabic language take the root as an invariant unit. Various authors, such as Ratcliffe and Ferrando, draw attention to this problem, stating that the concept of root, which may be reasonable in a synchronic analysis, tends to be inadequately driven to the field of historical derivation. In the words of Ratcliffe, "historically the notion of trilateral root is simply a convention of Arabic lexicography. It is the principle upon which dictionaries are organized" (Ratcliffe, 1998:13).

Recent research in the field of psycholinguistics, though, has brought into question the morphological and lexical status of the Semitic root—of course we mean from a synchronic perspective—as a relevant unit susceptible of segmentation in Arabic

words. An alternative hypothesis to the root-morpheme theory, which has been encouraged by recent authors (Boudelaa & Marslen-Wilson, 2001; Bohas, 2006; Bohas & Saguer, 2007), states that two unordered consonants of the stem bear the lexical content of an Arabic word. These two consonants, known as etymon, are nevertheless not treated as a morphological unit. "The etymon is defined as a non linearly-ordered bi-consonantal base made up of two phonemes taken from a given matrix" (Vesteegh et al., 2007) Since a typical Arabic word has three non-inflectional consonants, the left-over one is considered to be a meaningless extensional affix, i.e. a mere epenthetic consonant to the etymon. For example, two words such as وجل *wajal* and جمل *jamal*, are said to share the same etymon, جل *jl*.

	word 1	word 2	Etymon
Arabic	وجل	جمل	ل ج
Transliteration	wajal	jamal	j l
Translation	fear	camel	

FIGURE 14 Example of etymon

This idea comes from the fact that there seems to be several semantically related words which either do not share the same order of radicals, or do not share one of those radicals—the related words just have two radicals in common. Indeed, medieval Arab scholars already noticed this phenomenon and documented it extensively. Ibn Jinni, a grammarian and important phonetician from the 10th century, addressed this issue in his book "الخصائص" ("The special features"), a sort of encyclopedia about the linguistic science (Versteegh, 1997).

One of the topics analyzed in this work is what Ibn Jinni calls "اشتقاق الكبير"—"The great etymology"—which consists of the permutation of the consonants of triliteral roots—leading to six possible roots—looking for other existing roots with (at least) a vaguely related meaning. The concept of the etymon lied in the belief that the common semantic connotation is implicit in the shared radicals.

(34)

	word 1	word 2	Radicals
Arabic	عرب	عبرة	ع ر ب
Transliteration	çarab	çabraö	ç r b
Translation	nomads	tear	

FIGURE 15 Example of The great etymology
Taken from Versteegh (2001)

Another interesting procedure studied by the classical Arab scholars was the so-called "اشتقاق الأكبر"—"the greatest etymology"—which states that when two roots have an identical pair of radicals, independent of the third one, they share a semantic connotation (Versteegh, 2001). This, obviously, has a direct link with the etymon theory.

(35)

	root 1	root 2	Shared radicals
Arabic	فرر	فرق	ف ر
Transliteration	farara	faraqa	f r
Translation	to flee	to tear apart	

FIGURE 16 Example of The greatest etymology
Taken from Versteegh (2001)

However, contrary to the etymon hypothesis, recent psycholinguistic experiments suggest that the root-morpheme is a delimited unit. Idrissi and Kehayia exposed the main ideas of both hypotheses—the root-morpheme and the etymon morpheme—and the various experiments conducted on the topic (Idrissi & Kehayia, 2004). In general, evidence comes from studies with patients suffering from dyslexia. Dyslexic patients show processes of random metathesis which target root consonants and exclude affixal and epenthetic segments, as well as vowels.

Furthermore, Danks mentioned some research focused on the status of the root in involuntary slips-of-the-tongue, word games present in some Arabic dialects (McCarthy, 1981), and the formation of hypocoristics (Zawaydeh and Davis, 1999) and diminutives. They all showed processes of metathesis restricted to the root consonants, regardless of their position within the syllable, (Danks, 2011). As McCarthy stated, this is "another argument which supports the notion that the root consonantism is a single unit at some level of representation" (McCarthy, 1981).

Apart from cognitive experiments, the root-morpheme hypothesis has been encouraged by the OCP-Place restriction we described in Introduction-3.3. The OCP-Place stated that homorganic consonants are in general avoided within a root (Alderete et al., 2010). All this evidence are is to provide further arguments in support of the psycholinguistic reality of the root.

Some roots contain identical segments in second and third positions. These are the so-called geminate or doubled roots, e.g. ردد √rdd meaning 'replying, returning'. On the other hand, roots with identical first and second radicals are simply nonexistent[47]. Some authors[48] supported that these roots contain just two segments in underlying representation—consequently, under this definition they are termed biliteral or biconsonantal roots.

On the contrary, other authors refuse to accept a separate class of biliteral root. Danks argues that the existence of triliteral roots with identical second and third radicals may be well interpreted from a diachronic perspective[49]. He further explains that the identification of these roots as two-consonantal is contrary to Arabic tradition—which

[47] Apart from incidental examples—as the verb يَيَّ from a three equal consonants root √يىي, meaning 'to say the letter ya'—there are not examples of this sort in the Arabic lexicon.

[48] Among them, McCarthy and Prince (1990) stand.

[49] He cites the work of Al-Qahtani on diachronic analyses about phonological processes in Arabic.

invariably considers them as triliteral—and comes from the error of identifying the citation form of the simple verb—that corresponds to perf-3SgMas, e.g. مدّ *md*~—with the root of the verb—which in our example is √مدد √*mdd*. (Danks, 2011)

In relation to quadriliteral roots, there are cases of roots in which a biliteral segment has reduplicated itself to fit in a template of four radicals. The roots following this reduplication strategy are usually semantically marked; in many cases they express acts with an onomatopoeic referent, e.g. قهقه √*qhqh*, meaning 'giggling'. As Pierrehumbert claims, "almost all (roots with more than three consonants) are either reduplicative or conspicuously nonnative" (Pierrehumbert, 1992)

It is also important to notice that the only common element of a group of derived nouns and verbs is the discontinuous root string—the root is the only invariant unit present in an entire set of derivationally related forms.

4.4.2.2 Prosodic templates

In his 1981 article, McCarthy proposed a CV-scheme classification of the inventory of canonical pattern—C denotes non-head segments, including glides—with 8 types of templates:

1. CV.CVC
2. CVC.CVC
3. CVV.CVC
4. CV.CVC.CVC
5. CV.CVV.CVC
6. C.CV.CVC
7. C.CVC.CVC
8. C.CVV.CVC

He noticed that the templates presented some obvious regularities (McCarthy, 1981:386):

1. All templates end in a close syllable CVC
2. No template contains a sequence of two light syllables like CV.CV.CVC
3. No template contains a light syllable after a heavy syllable like CVC.CV.CVC
4. No template which begins with a consonant cluster is three or more syllables long overall

He claimed that to express these regularities grammar makes use of prosodic principles, which he represents conventionally through a CV structure; C represents the feature [segmental] and V [syllabic] (McCarthy, 1981).

In 1993, he changed his hypothesis and stated that there is just a single prosodic template, the one corresponding to pattern I. "The other measures are derived from this base by affixation. Therefore there are no separate templates for the other measures; all derive ultimately from the prosodically well-formed *fa.çal* template of Measure I" (McCarthy, 1993). This enabled him to apply moraic analysis to Arabic verbs (Beesley 1998a, 1998b; Kiraz, 1996)

4.4.2.3 The consonantal affixes

Below the derivational affixes and processes present in Arabic verbal stems are listed:

I	any of the form I patterns have derivational additions to the root
II	the second radical is geminated
III	the first vowel is lengthened
IV	*c* is prefixed

V *tV*[50] is prefixed and the second radical is geminated

VI *tV* is prefixed and the first vowel is lengthened

VII *n* is prefixed

VIII *t* is infixed between the first and the second radicals

IX the third radical is geminated

X *st* is prefixed

XI the first vowel is lengthen and the third radical is geminated

XII the second radical is geminated and *w* is infixed between the geminated radical

XIII a geminated *w* is infixed between the second and third radicals

XIV *n* is infixed between second and third radical and third radical is geminated

XV *n* is infixed between second and third radical and second vowel is lengthened

QI there are no derivational additions to the root

QII *tV* is prefixed

QIII *n* is infixed between the second and the third radicals

QIV the fourth radical is geminated

According to McCarthy, affixational analysis (1993) described in the previous section, the derivational material of the derived patterns is added to the pattern I template, the single prosodic template present in verbal grammar. The derivational processes, and their effect on the CV template, are interpreted by means of prosodic notions. He described some of the pattern affixes.

Some pattern II verbs are reported to be the result of a reanalysis process: geminated imperfective forms were metanalyzed as pattern II verbs[51]. Corriente reported a list of these intensive pattern forms from Andalusian Arabic which were reanalyzed as pattern II verbs (Corriente, 2004).

[50] V represents a vowel.

[51] Akkadian inflectional system included a gemination mark for the imperfective tense.

Pattern III presents a lengthening of the initial vowel (Danks, 2011); it adds a mora to the initial syllable of the pattern I template, the base form. Also starting from pattern I, pattern II geminated the second radical, but in this case by prefixing a mora under negative prosodic circumscription (McCarthy, 1993). As a result of these processes, both pattern II and III have a heavy or bimoraic first syllable. Quantitatively, they are identical, even though being qualitatively different (Danks, 2011)

Patterns V, VI and QII are said to derive from patterns II, III and QI respectively by means of the tV- prefix (McCarthy, 1993; Danks, 2011). Danks, as we already pointed out, defined it as a detransitivising affix. It is noteworthy that it is the only derivational affix formed by a full syllable. Both patterns IV and X also present prefixes, but cause the deletion of the medial vowel of the underlying form (McCarthy, 1993). For instance, form X *stafçal* from *stafaçal.*

McCarthy claims that the infix -t- of pattern VIII is prefixed to pattern I template under negative prosodic circumscription of the initial consonants (McCarthy, 1993). Historically, "Form VIII [is interpreted] as having a prefix t- that gets moved subsequently by a controversial 't-flop' rule after the first radical" (Beesley, 1999).

Pattern QI is considered atemplatic by McCarthy (1993). Interestingly, it presents many roots formed by geminated two-consonant sets, usually conveying onomatopoeic content referring to sound (Ryding, 2005). A fair deal of Pattern QI and QII verbs are used to introduce borrowed words from other languages.

4.4.2.4 Vocalic melody morpheme

In the following table we include a representation of the vocalism of each pattern—for the sake of convenience, we represent the vocalic string linearly even though it is a non-concatenative element. The vowel of the prefix ta- is not represented.

Pattern		Vocalization		
		p-stem[52]	i-stem	m-stem
triliteral	Iau	aa	au	u
	Iai	aa	ai	i
	Iaa	aa	aa	a
	Iuu	au	au	u
	Iia	ai	aa	a
	Iii	ai	ai	i
	II	aa	uai	ai
	III	aa	uai	ai
	IV	aa	ui	i
	V	aa	aaa	aa
	VI	aa	aaa	aa
	VII	aa	aai	ai
	VIII	aa	aai	ai
	IX	aa	aa	a
	X	aa	aai	ai
	XI	aa	aai	ai
	XII	aa	aai	ai
	XIII	aa	aai	ai
	XIV	aa	aai	ai
	XV	aa	aai	ai
quadriliteral	I	aa	uai	ai
	II	aa	aaa	aa
	QIII	aa	aai	ai
	QIV	aa	aai	ai

TABLE 10 Vocalism morphemes

[52] Recall that *p-stem* stands for perfective stem, *i-stem* for imperfective stem and *m-stem* for imperative stem.

Many authors have acknowledged the fact that there seems to be a subtle regularity in the organization of the vocalic qualities through the different patterns, both in the perfective and imperfective verbal paradigms. In the following lines we present some of these authors and their claims.

As McCarthy stated, "the vocalism—what I call the vowel melody—is not freely distributed among the vowels" (McCarthy, 1981). And he further remarks that "some vowel patterns seem to bear consistent meaning" (McCarthy, 1981). For instance, looking at the last vowel of the pattern stems, we notice a tendency of perfective stems exhibiting mostly a vowel *a*, whereas in the imperfective the vowel tends to be *i*. However, the interpretation of the vowels' semantic and morphosyntactic properties has not been completely clarified yet.

Ryding states cautiously that "there are shades of meaning associated with the stem vowel differences in the past tense citation forms, but these semantic differences are very subtle" (Ryding, 2005:455). Danks, following Wright and Badawi's view, points out that the vocalic material may be explained on the grounds of a purely lexical basis: "While there is clearly some systematicity relating the p-stem [imperfective] vowel to its s-stem [perfective] counterpart, opinions differ as to whether there is good synchronic evidence of syntactic or semantic consistency to the different vowelling schemes, suggesting a meaningful classification on this basis as per Wright (1967:I.30), or whether, as Badawi et al. (2004:60) claim, they are "best treated as a lexical feature" (Danks, 2011:18).

In relation to pattern I verbs, Danks adds that the middle vowel of the p-stem to some extent determines the quality of the i-stem vowel and suggests considering the p-stem vowels as non-morphemic (Danks,2011:45). However, he goes on to say that Ratcliffe, in turn, considers the i-stem to be basic over the p-stem. Actually, from a diachronic point of view it has been proved that the perfective aspect is a later formation. In this

regard, Ferrando (1999:11) states that "El perfectivo es en las lenguas semíticas una formación posterior a partir de un esquema nominal, como demuestran los hechos del acadio"[53]. This may not be the case synchronically, yet the contrary cannot be claimed either.

Badawi, in turn, offers a more compact justification in support of the lack of morphological content of the vocalic melody string: "the vowels vary in both aspects, originally perhaps on the grounds of verb class (stative, transitive, intransitive, etc.), but now best treated as a lexical feature: while it is true that all verbs with -u- as their medial vowel in the perf. are intransitive and also have -u- in the imperf., the *yaf'ulu* form may be transitive. Even the crude generalization that most verbs are of the pattern fa'ala in the perf. and yaf'ulu in the imperf. is unreliable" (Badawi et al., 2004:60). In the same fashion, Danks points out the opinion of Holes in his work "Modern Arabic: structures, functions and varieties" (2004:101), who considers that the meaning of the second vowel in pattern I conveys transitivity and dynamic versus stative (Danks, 2011:45).

Another remarkable issue is the tendency that p-stems of Pattern I with *i* or *u* as their second vowel have transitive and stative meaning. The Iuu class is semantically determined for stativity or quality; many of the verbs belonging to this class are deadjectival verbs. Verbs of the pattern Iia, in turn, are often intransitive or stative, however, there are several exceptions (McCarthy, 1994).

Pattern Iaa has a special relevance for it is said to be phonologically determined—e.g. verb فَتَحَ *fataHa* 'to open', perfective فَتَحَ *fataHa* "(he) opened" – imperfective يَفْتَحُ *yaf·taHu* '(he) opens'. Various authors (Versteegh et al., 2007; Wright, 2007; McCarthy, 1994; Danks, 2011; Watson,2007, among others) have reported the fact that guttural consonants—recall that the guttural class was composed of uvulars خ *x* and غ *g*,

[53] "perfective is in Semitic languages a later formation from a nominal pattern, as shown by the evidences in Akkadian".

pharyngeals ح *H* and ع *ç* and lagyngeals ء *c* and ه *h*—tend to lower adjacent vowels due to their pharyngeal characteristic. As a result, many *a* vowels are derived from assimilated *i* or *u*. In this sense, Brame (1970) and later McCarthy claimed—and McCarthy endorsed this idea with data from the Hans Wehr dictionary—that pattern Iaa does not represent a real conjugational class, but it corresponds to Iai or Iau patterns whose second and third radicals are guttural consonants. In fact the idea of an assimilation process in verbs of the type Iaa can be traced back to Sibawayhi as well as to a later work by Ibn Jinni known, collected in his book "الخصائص" ("The Characteristics"). The fatha vowel which results from the assimilation is referred to الإدغام الأصغر 'the partial assimilation' by Ibn Jinni (Mas'ūd and Åkesson, 2001).

In relation to the semantics of pattern I different vocalizations, Dichy (2007) claimed that perfectives of the form *façala* are mainly transitive, though a good number of verbs from this pattern are intransitive. The form *façila*, he adds, have an equal amount of transitive and intransitive representatives. On the contrary, *façula* verbs are always intransitive.

Regarding the relationship established between perfective and imperfective vowel melodies in pattern I, McCarthy states that they are not marked by an apophonic relationship, i.e., alterations on vowel quality in the vocalic string do not imply affixal distinctions. "It is further clear that there is no unambiguous ablaut function from perfective to imperfective or vice versa. That is, given any vowel in one aspect, we cannot uniquely determine its quality in the other aspect" (McCarthy, 1981). Yet, he claims that if we exclude the pattern Iuu—which is semantically marked by stativity—it is possible to relate imperfective to perfective; There is a polarity shift between imperfective, marked by [+high], and perfective [-high] (McCarthy, 1981). Cahill (2010), as well, supports the existence of this ablaut process operation in the vocalic discontinuous string of Arabic verbs.

Ferrando remarks the importance this polarity shift has in McCarthy's theoretical framework. He described his redesign of the root-and-pattern model by saying that McCarthy's analysis "consiste en un mayor nivel de abstracción que el proporcionado por el tradicional análisis "raíz + esquema", y que se basa en la observación del material prosódico representado en fórmulas o *templates* y en el importante principio de polaridad vocálica, según el cual es el juego de oposiciones vocálicas, básicamente vocales de formante alto (i, u) frente a vocales de formante bajo (a), lo que permite explicar satisfactoriamente varias de las estructuras morfológicas del árabe (formación de verbos no agentivos, derivaciones verbales y, naturalmente, pluralización) mediante reglas sencillas" (Ferrando, 1999:11)[54].

4.4.3 Inflectional morphology

Inflectional morphemes represent the morphosyntactic relations of the constituents of the language, so they have a purely linguistic meaning. Arabic inflectional categories and their values are listed below (revised in Ryding, 2005; Wright, 2007; Bahloul, 2008; Al-Najem, 2007; Cowan, 1958; Haywood & Nahmad, 1962):

1. Voice
    a. Active معلوم
    b. Passive مجهول

[54] McCarthy's analysis "consists of a higher level of abstraction than the one provided by the traditional root-and-pattern approach. The analysis is based on the prosodic material, represented by means of templates and on the important principle of vowel polarity shift. This polarity shift mainly alternates vowel oppositions of high formant vowels (i,u) against low formant vowels (a). This principle allows us to explain satisfactorily some of the Arabic morphological structures (formation of non-agentive verbs, verbal derivations and, obviously, pluralization) though simple rules"

2. Aspect/tense
    a. Perfective/past ماضي
    b. Imperfective/present مضارع/حاضر

3. Mood
    a. Indicative مرفوع
    b. Subjunctive منصوب
    c. Jussive مجزوم
    d. Imperative أمر

4. Person
    a. First متكلّم
    b. Second مخاطب
    c. Third غائب

5. Number
    a. Singular مفرد
    b. Dual مثنّى
    c. Plural جمع

6. Gender
    a. Masculine مذكّر
    b. Feminine مؤنّث

4.4.3.1 Voice

We quote here the definition of voice provided by Ryding: "whereas the tense of a verb conveys temporal or time-related information, the *voice* of a verb conveys information on the topical focus of a sentence. The active voice is used when the doer

of an action is the subject of the verb, and the passive voice when the object of a verbal action is the subject" (Ryding, 2005: 445)

The passive voice is formed by changing the p-stem vowel string into *ui* and the i-stem into *ua*. The active is the unmarked feature and renders its form to the specific vocalism of each pattern. The characteristic vowel of the passive vocalism is therefore the *u*, as it is found in both p-stem and i-stem vocalisms.

p-stem	*ui*
i-stem	*ua*

All verbal patterns are susceptible to forming a passive counterpart from the unmarked active by the apophonic—or ablaut—process represented below. This internal passive is only restricted when the verb is semantically incompatible. Wright states that the following patterns do not form the passive due to semantic restrictions: Iuu patterns, patterns from IX to XI excluding X, and the rest of patterns I if they designate not an act—transitive or intransitive—but a state or condition (Wright, 2007:49).

On the passive formation Danks states: "(...) I tentatively conclude that actional passive formation is most likely a matter of the semantics of the individual verb, rather than a formal property of the pattern themselves." (Danks, 2011:241) However, we have to add that in real usage the inflectional passive is not very productive.

4.4.3.2 Aspect and tense

Depending on the sources, Arabic verbs are said to be marked for aspect or for tense. The former presents the aspectual distinction perfective versus imperfective, while the latter opposes past and present tense. Ryding offers an explanation of this situation by claiming that "The difference between tense and aspect can be subtle, and

the two categories may overlap to a significant extent" (Ryding, 2005:440). Maher Bahloul studied this problem in an extensive research (Bahloul, 2008). He classifies all the previous hypotheses throughout Arabic linguistics in four categories:

a. Aspectual hypothesis[55]
b. Temporal hypothesis[56]
c. Aspectual-temporal hypothesis[57]
d. Non-aspectual nor temporal hypothesis[58]

The temporal hypothesis was first proposed by Sibawayhi (ca. 760, ca. 796), the author of the first Arabic grammar and one of the most influential works in the field, "الكتاب" ("The book)". This is the reason why the temporal hypothesis has been the most accepted in the Arabic linguistic tradition. Sibawayhi described a basic opposition in the Arabic verbal system of past-present-imperative.

Bahloul uses a written corpus to analyze real examples of the language so that he can systematize the semantic values of Arabic verbs in an empirical way. The corpus consists of texts from the news, from the academic field and from literature (13 newspaper articles, 5 academic articles and 5 contemporary short stories)[59]. Based on the various examples found in the corpus, Bahloul carries out an analysis of the semantic properties of the Arabic verbal morphology. Thus, he categorizes the

[55] The main advocates of the theory are Wright, Jusmanov, Blachère, Cohen, Fleisch, Beeston, McCarus, and Al-Mansouri.

[56] The main advocates of the theory are Sibawayhi, ash-Shirbiinii, Aartun, Benmamoun, Wightwick and Gaafar, Banat.

[57] The main advocates of the theory are Comrie, Messaoudi, Fassi Fehri, Fischer, and Baterson.

[58] The main advocates of the theory are Kurylowicz, Schulz.

[59] Bahloul does not specify the total number of words in the corpus. The 13 newspaper articles are gathered in his book (Bahloul, 2008).

semantic features expressed by the verbs. He states that the perfective can be used in these cases:

a. Past time (without specifying the distance in time)
b. Present time
c. Atemporal expressions
d. Future time[60]

As noted by Bahloul, the fact that the perfective may be used for the present, as well as for atemporal expressions, shows that it is not restrictively attached to the past tense. However, it does not mean that all the possible uses of the verb have the same importance (frequency of occurrence), nor are they used freely in any context. It is necessary to distinguish between primary uses and secondary uses. In fact, the more direct use of the perfective is the expression of past tense and, furthermore, all the remaining uses are contextually conditioned.

Consequently, Bahloul concludes that the perfective contains two inherent semantic values:

1. Anteriority, the event occurs before a specific moment.
2. Dimensionality, the event refers to an interval and its dimensions are concrete and defined.

The imperfective includes the following uses[61]:

a. Present time
b. Atemporal expressions

[60] For instance, in conditional constructions such as the ones with the particle اذا

[61] Note that they are the same as in the perfective.

c. Future time
d. Past time

First of all, it must be noted that the particles of verbal negation, with the exception of ما always need the imperfective to appear. Temporality is marked by the particle itself: لا for the present, لم for the past and لن for the future. This demonstrates that the imperfective is not clearly marked for temporality. On the other hand, the temporal particle to express future, سوف or -س in its cliticized form, only works with the imperfective, as in the previous examples. In addition, a bare imperfective verb may be used for constructions where the future tense is lexically explicit.

Therefore, we must state that the imperfective does not show any restrictions in temporality. In this case, temporality is an inherent property of the linguistic context. Due to this lack of temporal characterization, it is normal to find imperfective verbs in contexts situated in the past. It appears that the perfective is the marked form within the Arabic verbal system, whereas the imperfective is the non-marked form. As a result, the semantic features attached to the perfective are those of anteriority and dimensionality, whereas the imperfective implies all of which the perfective does not cover, i.e. all the features which oppose that. Therefore, the categories are as follows:

a.	Perfective	+ anterior
		+ dimensional
b.	Imperfective	- anterior
		- dimensional
		± anteriority

± dimensionality

Formally, The perfective/imperfective opposition is expressed by means of two procedures:

1. the vocalism string: different vowel melodies are related to different tense/aspects.

2. the inflectional paradigm: each conjugation paradigm, i.e. the group of inflectional affixes carrying the information of mood, person, number and gender, is specialized for one of the two aspects. Thus, a perfective stem binds to one set of inflectional affixes and an imperfective stem binds to another set of inflectional affixes, which in turn differentiate indicative, subjunctive and jussive moods.

Inflec. info	perfective (only suffixes)	imperfective indicative	imperfective subjunctive	imperfective jussive
3SM	-a	y-u	y-a	y-
3SF	-at	t-u	t-a	t-
3DM	-aA	y-aAni	y-aA	y-aA
3DF	-ataA	t-aAni	t-aA	t-aA
3PM	-uwA	y-uwna	y-uwA	y-uwA
3PF	-na	y-na	y-na	y-na
2SM	-ta	t-u	t-a	t-
2SF	-ti	t-u	t-a	t-
2DN	-tumaA	t-aAni	t-aA	t-aA
2PM	-tum	t-uwna	t-uwA	t-uwA
2PF	-tun~a	t-na	t-na	t-na
1SN	-tu	Â-u	Â-a	Â-
1PN	-na	n-u	n-a	n-

TABLE 11 Affixes of the inflectional paradigm

The final *A* of the inflectional form 3PM has an orthographic function, it is not pronounced and has no linguistic content.

4.4.3.3 Mood

The mood of the verb indicates the degree of reality and the conditions under which the act expressed by the verb is implicated. Arabic verbal system opposes four moods, but they only interact with the imperfective inflection; the imperative mood is derived from the jussive:

1. indicative mode involves real statements;

2. subjunctive mode involves desiderative or hypothetical acts;
3. jussive involves commands;
4. imperative involves commands too.

Due to their high syntactic value, mood morphemes are attached at the most external extremes of words, they are the last suffixes. For all singular forms (except the second person feminine), the suffixes are systematically –u for indicative, -a for subjunctive and lack of suffix for jussive—a sukun will be added to the jussive forms which end in consonants.

Nom	**Subj**	**Jussive**
-u	-a	None

Additionally, the indicative has two more suffixes for other forms:

-na	feminine singular second and third persons plural
-ni	second and third persons dual

The suffix -A in inflectional forms 2PM and 3PM of subjunctive and jussive is orthographical.

Imperative differs from nominative, subjunctive and jussive in not having prefixes. This is in fact proof that imperfective prefixes are only marking person; all imperative forms are in second person, so no distinction is necessary for person. The suffixes are the same as the jussive mood.

4.4.3.4 Person

The feature person marks the relation between the agent of the action and the speaker. Arabic follows a typical distinction of three possibilities:

a. first person, the agent is the speaker;
b. second person, the agent is the listener;
c. third person, the agent is neither the speaker nor the listener.

In relation to the morphemic material, it is difficult to separate person, number and gender features in the affixational paradigm. In both inflectional paradigms, morpheme boundaries cannot be clearly defined. There are some clues though: in the perfective stem, the third person is marked by means of the infix -a- (except 3PM), the second person is -tum or -tun, and first person is -tu for singular and -n is characteristic for plural. In the imperfective, the prefix y- marks the third person (except for form 3SF) and t- the second person; first person has again distinct forms according to number, Á- for singular and n- for plural—which is the same segment as in the perfective and, moreover, it is shared by the plural suffix -uwn used in verbal and nominal inflection.

p-stem paradigm

3rd person: -a-
2nd person: -tum / -tun
1st person: -tu / -n

i-stem paradigm

3rd person: y-
2nd person: t-
1st person: -Á / -n

4.4.3.5 Number

The grammatical number typically derives from the referential content of the statement. When more than one entity (concrete or abstract) is involved in the action, this is expressed grammatically by a plurality feature. Arabic verbs distinguish three numbers:

a. singular;
b. dual—exclusive, it only exists for second and third person;
c. plural.

However, the grammatical number sometimes shows non-coincidental relations with the entities they refer to in the real world. Nouns expressing non-rational entities, when pluralized, show agreement in singular feminine with verbs and modifiers.

(36)

Arabic	اللغات تتطوّر
Transliteration	Al-lugaAtu tataTaw~aru
Translation	Languages (pl.) evolve(f. sg.)

Formally, number is marked in the verbal paradigm as follows: in the p-stem and in i-stem the singular is the unmarked feature, so there is additional material for dual and plural; dual is marked by a suffix -A—as in nominal inflection—and plural by the suffix -w—again as in nominal inflection—the first person do not use these suffixes. feminine plural is mostly marked by the suffix -na.

In both p-stem and i-stem:

dual	-A
plural	-w
feminine plural	-na

4.4.3.6 Gender

The grammatical gender is a kind of nominal classifier. The gender system of Arabic presents the typical dichotomous distinction:

a. masculine
b. feminine

The unmarked feature is the masculine gender; the most relevant characteristic of the feminine is the segment *t* in both p-stem and i-stem. Additionally, feminine singular of the i-stem defined by the suffix -iy. As we already stated in the previous section, the suffix -na is bound to feminine plural feature.

Formally, the features of person, number and gender are partially merged in the affixational material.

5 State-of-art in Arabic computational morphology

5.1 The development of natural language processing

The aim of Natural Language Processing (NLP) is essentially to find a suitable formal representation of natural language which enables the interaction between humans and machines. At present, there are two main approaches which address the field: symbolic models and stochastic models (revised in Joshi, 1987; O'Grady et al., 1996; Moreno Sandoval, 2001; 2009; Hausser, 2001; Hauser, 2001; Roark & Sproat, 2007; Jurafsky & Martin, 2009).

The symbolic models for representing computationally natural language have their origins in the generative grammar and the Chomskyan school. The generative grammar started to describe language as a formal system: each language consists of a grammar—a well-defined set of rules—and a lexicon—axiomatic elements at the service of rules. Language structure was seen as a finite set of elements being able to represent infinite sentences, thus language is described simply as a sequence of symbols (Jurafsky and Martin, 2009). Consequently, rules are said to be applied in a recursive way, i.e. a structure may contain itself in a potentially infinite number of times. Recursion is an essential property of every natural language.

The first computational devices accepted by Chomsky and his colleagues to codify formal languages were finite-state machines in the form of finite-state transducers (FSTs). In computational linguistics, FSTs are powerful and precise mechanisms used for representing some kind of linguistic rewrite rules—free context rules. FSTs establish operations of mapping from one set of symbols to another. This technology is usually very efficient for dealing with phonological and concatenative

morphological processes—especially inflection—and additionally with orthographic variations, providing linguistic analysis with mathematical foundation. FSTs describing natural language processes are usually specified under the form of regular expressions. A FST is commonly run in a *cascade*, i.e. sequentially.

From this moment on, most linguistic theories aimed at defining formalisms to represent natural languages. Those formal languages, in turn, were effectively represented by means of computational systems. From the mid-sixties until the mid-seventies, the dominant theory was that of transformational grammar. The phrase structure grammar defined by the early generativism was unable to represent effectively natural language, hence a new mechanism was developed: the transformations.

The syntactic relations amongst the constituents of the sentence were considered primitive elements. These primitive elements were then manipulated through transformational processes to convert the deep level structures to surface level elements. The strong point of the transformational approach was focused on the network of rules, whereas the lexicon was quite simple. The rules were arranged using the so-called augmented transition networks.

The transformational formalism had several problems though. The rules were extremely complex and difficult to define. The systems hence lacked computational efficiency. In the late seventies, the field was looking for a more powerful formalism which avoided the disadvantages of the transformational approach. The new mechanism found was unification.

The new model, known as Unification Grammar or Constraint-base Grammar, was widely used and developed in the eighties (Sag et al., 1986). The formalism was in fact less powerful, yet proved to be much more efficient in computational applications. Unification grammar tries to describe natural language through the use of feature

structures. “Feature structures are simply sets of feature-pairs, where features are unanalyzable atomic symbols drawn from some finite set, and values are either atomic symbols or feature structures” (Jurafsky and Martin, 2009:393). The lexicon is first annotated with linguistic information in the form of these feature structures. Then all that information is combined and unified using a mechanism called unification. Unification consists of merging all the feature information into a global and coherent structure, which is used to describe the grammar. As a result, the lexicon becomes more important, compared to the rule component. However, too extensive lexicons tend to saturate the information collected in the feature structures and consequently are less efficient. The best solution is to find a balance in the relation between the lexicon and the rules.

Unification is indeed an extremely powerful mechanism used by several grammatical theories. Amongst the most prominent theories that emerged under the scope of the unification principles, are Generalized Phrase Structure Grammar (GPSG), Head-driven Phrase Structure Grammar (HPSG), Lexical Functional Grammar (LFG), Categorial Grammar (CG) and, more recently, Dependency Grammar (DG).

Perhaps, the most significant aspect of the current linguistic theories based on the Unification Grammar is that, as Joshi stated, “any reasonable grammatical formalism can be instantiated in the unification formalism” (Joshi, 1987:47). This gives an idea of the power the unification mechanism has.

Transformational mechanism	vs.	Unification mechanism
complex rules		simple rules
simple lexicon		complex lexicon
rewrite rules		feature structures
less computational efficiency		more computational efficiency

TABLE 12 Approaches to symbolic models: Transformation versus Unification

With regards to morphology, one of the most successful formalisms was the so-called Two-Level morphology, which makes extensive use of FST devices. Two-level morphology is a powerful and efficient formalism for dealing with linguistic analysis and generation—especially morphophonological phenomena—at a time. It was mainly developed by Kimmo Koskenniemi, Lauri Kartunnen, Ronald Kaplan and Martin Kay. The Two-level model proposes a direct relation between the lexical level and the surface level, representing both levels as a concatenation of string elements. Each element of the lexical level is mapped with one element of the surface level, so there is a bidirectional representation of each of the elements. Therefore, two-level morphology descriptions present the advantage of being suitable both for analysis or generation; further, they present a computationally robust and efficient model (Karttunen and Beesley, 2005).

FSTs representing a Two-level morphology model make use of regular expressions, a powerful language highly suitable for formalizing rewrite rules. As Harald Trost noted, "because most morphological phenomena can be described with regular expressions, the use of finite-state techniques for morphological components is common" (Trost, 2005:39). Kay (1987) explains that finite-state and two-level morphology were initially built to formalize autosegmental phonology proposed by Goldsmith and autosegmental morphology proposed by McCarthy (Al-Sughaiyer and Al-Kharashi, 2004).

It is worth noting that all grammatical formalisms are indeed partial theories that attempt to account for natural languages. There is no system capable of explaining all the linguistic phenomena of a language.

Unification Grammars, as well as Transformational Grammars, are symbolic models, based on logic principles, which attempt to give a precise description of natural language. In the nineties the reliability of symbolic models was put into question. Stochastic paradigms, based on quantitative models began to be used due to their effectiveness; they are indeed less precise, yet more robust and easily implemented. Furthermore, and contrary to symbolic models, they proved to be able to deal with linguistic ambiguity, because they can infer statistical regularities from corpora of data (Moreno Sandoval, 2001).

The maximum power of Statistical Models seems to have been achieved. Recently, symbolic models have been reemerging. The aim nowadays is to find a well-balanced combination of both models.

Symbolic models	vs.	Statistical models
More powerful		Less powerful
Less efficient		More efficient

TABLE 13 Symbolic models versus statistical models

In a nutshell, the analysis of natural language by means of formal operations has influenced the different paths taken by modern linguistic studies. In the words of Cooper and Ranta (2008): "The view of natural languages as formal languages was a tremendously productive abstraction which enabled us to apply twentieth century logical techniques to the characterization of human ability".

5.2 Analytical strategies[62]

A morphological analyzer is a computational tool which, given a word as an input, returns all possible analysis of that word without regarding the context. This means that a morphological analyzer needs to create a language model which best describes that language. Sometimes, they may be used both for generation and analysis, in which case they are said to be bidirectional.

Morphological analyzers are composed of two basic parts (Kiraz, 2001):

1. A lexicon of words (lexical units), responsible for the coverage of the system. Ideally, the lexicon should include all the morphemes of a given language. The lexicon must contain the maximum amount of grammatical information; as Dichy states, "grammar-lexis relations are an essential part of any Arabic lexical resources" (Dichy, 2002).
2. A set of linguistic rules (morphosyntactic knowledge), responsible for the robustness of the system. There are mainly two types of rules:
    a. rewrite rules, which handle the phonological and orthographic variations of the lexical items,
    b. morphotactic rules, which determine how morphemes are combined.

In fact both the lexicon and the rule components are closely related: some linguistic rules can be codified in the lexicon, and consequently the size of both parts is directly related.

[62] Revised in the article "Arabic Computational Morphology: Knowledge-based and Empirical Methods" published in Soudi et al. 2007; Attia et al., 2011; Al-Sughaiyer et al. 2004, Habash, 2010; and Kiraz, 2001.

Morphological analyzers, as a tool, are essential components of more complicated systems used in artificial intelligence, automatic translation, speech recognition systems, etc. The aim of NLP is to find the most efficient way to describe formally a language for a specific application (serving as human-machine interface).

Most of the Arabic morphological analyzers have been built according to a knowledge-based paradigm instead of an empirical model—i.e. they follow a symbolic instead of a statistical approach. Within the knowledge-based paradigms, the majority of the systems are based on a root-and-pattern analysis in one way or another. Yet they follow different strategies depending on the degree of abstraction suggested by the analysis. The main strategies of analysis are the following:

a. **Syllable-based**: the analysis is based on syllable structure. This analytical paradigm was in fact developed to describe European languages—especially the ablaut processes encountered in Germanic languages. Lynne Cahill (2007, 2010) applied the syllable-based model to Arabic grammar. This paradigm does not follow a strict root-and-pattern model as described by McCarthy, yet it organizes the lexicon in sets of roots, patterns and vowel inflections. The main advantage of this paradigm is that it attempts to construct a grammatical theory, in the sense that it uses the same descriptive architecture to describe different family languages[63].

b. **Root-based**: it typically follows a root-and-pattern approach á la McCarthy. Stems are formed by the combination of a root and a vowel melody, arranged according to canonical patterns. Soudi et al. (2007:5) remarks that "McCarthy's autosegmental approach is reflected in most of the computational attempts to model Arabic morphology, especially in the

[63] This is consistent with one of the objectives of morphological analysis we pointed out in section 4.1—to describe different languages under the same descriptive architecture, so that we can freely compare the linguistic structures of these languages amongst themselves.

systems written within finite-state morphology (Beesley, 1990, 1996; Kay, 1987; Kiraz, 1994a, 1994b, 2000)". The Xerox Arabic Morphological Analyzer (Beesley, 2001), developed on this approach, is probably the most striking example of this model. It is based on the interdigitation of a list of roots and a list of patterns to analyze Arabic words. The majority of the systems developed in recent years include a root-and-pattern representation, for instance Gridach and Chenfour (2011). Different applications such as diacritization systems and small morphological modules which are part of larger applications, often give a special treatment to the root as a lexical item (Rashwan et al., 2009; Dakkak et al. 2000; Kanaan et al., 2003)

c. **Lexeme-based**: lexical items are segmented into a lexeme and its inflectional features. This approach "supports the claim that the stem is the only morphologically relevant form of a lexeme" (Soudi et al., 2007:7). Thus, the lexeme-based model focuses on representing the stem and the alterations suffered at this level, and not at deeper levels. This model is consequently easier to develop. The most relevant models following this strategy are AraComLex, a morphological processing toolkit developed by Mohammed Attia (Attia et al., 2011a: 2011b), and Cavalli-Sforza et al. (2000). On the advantages of lexeme-based morphology, Attia states "We believe that a lemma-based morphology [i.e. lexeme-based morphology] is more economical than the stem-based morphology as it does not list all form variations and relies on generalized rules. It is also less complex than the root-based approach and less likely to overgenerate (Dichy and Farghaly, 2003; Attia, 2006). "This leads to better maintainability and scalability of our morphology" (Attia et al., 2011b).

d. **Stem-based**: In stem-based models alteration rules are codified directly in the lexicon; all variants of a lexeme are included as lexicon entries, so no modification rules are necessary. This technique requites less analytical

effort, but provides little linguistic consistency. The most famous stem-based analyzer for the Arabic language is the Standard Arabic Morphological Analyzer (SAMA)—formerly known as BAMA—developed by Tim Buckwalter (Buckwalter, 2004). It includes all the variants of each stem in the lexicon. The lexicons provided by SAMA have been redesigned in various projects. One of the most remarkable of this works is the ElixirFM morphological analyzer, developed by Otakar Smrž, which in turn follows an *ad hoc* root-and-pattern approach. Another system built on this paradigm is AraParse (Ouersighni, 2001).

On the computational side, the clearest approach to handling Semitic morphology is found in Kiraz (1994b, 2000, 2001). He indicated that morphotactic grammars can be implemented in two ways: using finite-state automata—especially for most purely concatenative languages—or unification-based context-free grammars (Kiraz, 2001:16). In the case of Arabic it is quite usual that the systems are implemented using FSTs, independently from the linguistic model they follow.

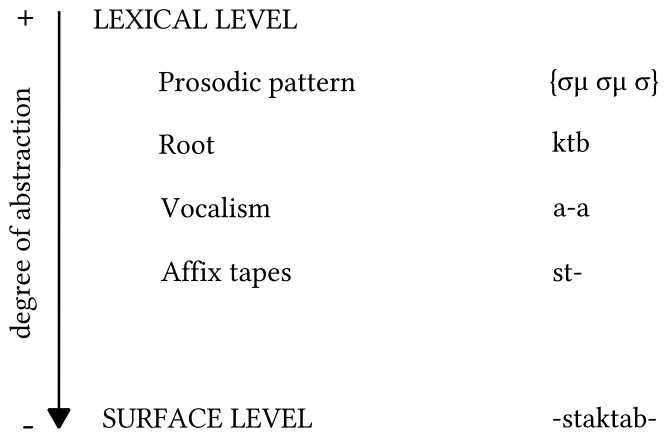
5.3 Survey of current morphological analyzers

Below, we are going to provide a brief description of the most relevant morphological analyzers or computational models carried out in the field of Arabic language processing. A comparison table is included at the end of the section.

5.3.1 Multi-Tape Two-Level Morphology Model (Kiraz)

George Kiraz (1994a; 1994b; 1996; 1999; 2000; 2001), following the work of Kay from 1987, developed and implemented a prosodic model for analyzing Arabic morphology. The model is based on the prosodic analysis of Arabic carried out by McCarthy and Prince. As for the implementation, Kiraz made use of an augmented version of Two-Level morphology, a Multi-tape representation of Arabic morphological system, compiled into FSTs. Multi-tape representations are extensions to two-level morphology which operate with the lexical level. The augmented Two-level morphology approach establishes a multi-tier representation of Arabic root-and-pattern morphology, based on moraic syllabic weight and infixation processes. The multi-tape representations correspond to four parallel levels: prosodic pattern, root, vocalism and affix tapes. The whole multi-tape process maps a set of lexical levels with a surface level. Below, we have an example of an Arabic stem representation:

(36)



The prosodic pattern is focused on the stem circumscription. The present example indicates that the stem is formed by three syllables, the first two bearing monomoraic weight and the third being dependent to the inflectional suffix which must be added to the stem. It does not exhibit moraic weight in the stem circumscription.

This is perhaps the most interesting model that has attempted to formalize computationally McCarthy and Prince analysis of Arabic morphology.

5.3.2 The Xerox Arabic Morphological Analyzer (Beesley)

Xerox (Beesley, 1998a, 1998b, 1998c, 2001) is a finite-state morphological analyzer for Arabic based on full-vocalized lexicon and rules. It was developed by Kenneth Beesley. It started to be created in 1996 at the Xerox Research Centre Europe, as a reimplementation of a previous work, a prototype analyzer called ALPNET (1991), which followed a two-level morphology paradigm. The lexicons of ALPNET were redesigned to fit this new proposal and they also included an English gloss for all the entries.

The lexicons are divided into four databases which include morphotactic coding and full vocalized lexical entries: prefixes, suffixes, roots and patterns. The system extracts the information stored in the lexicons and compiled it into a FST. The system allows overgeneration in a first step, but removes it at the following levels. Phonotactics and orthographic variation rules are also compiled into FSTs.

The lexicon includes 4,930 roots, all having been inserted by hand, and the subset of patterns corresponding to each of them. On average, each root can create 18 different lemmas, so the interrelationship between roots and patterns creates 90,000 lemmas. The combination of prefixes, stems and suffixes yield over 72 million hypothetical forms—with the disadvantage that it overgenerates. The phonotactic treatment includes 66 finite-state variation rules. The system is capable of analyzing words which are totally or partially vocalized. The aim of this project was to create a tool for pedagogical support and at the same time to be useful for bigger systems of natural language processing.

5.3.3 A lexeme-based model (Cavalli-Sforza et al.)

Cavalli-Sforza et al (2000) and Soudi et al. (2001) proposed a lexeme-based model for Arabic morphology based on concatenative strategies. The model is built on fully vocalized words. Cavalli-Sforza's approach is motivated by practical concerns—root-based models are more complex and thus less manageable and hard to develop. Cavalli-Sforza claims that a lexeme-based model reduces this complexity.

The model is focused on generation through concatenative procedures. The system uses the tool MORPHE (Levitt, 1992) for modelling morphology based on discrimination trees and regular expressions. Each internal node of the tree specifies a piece of the feature structure (FS) that is common to that entire subtree. FS contains elements composed of feature-value pairs. Complex values are themselves a FS.

To analyze irregular forms, the system keeps information on the middle radical and vowel of verbal lexemes so that the correct rules are applied to generate the inflection. The input of the MORPHE system consists of the FS indicated above, which describes the item the system has to transform. FSs are represented in recursive Lisp lists.

The system was evaluated on a subset of verbal morphology—regular and weak verbs, and nouns.

5.3.4 Standard Arabic Morphological Analyzer (Buckwalter)

The Standard Arabic Morphological Analyzer (SAMA), formerly known as Buckwalter Arabic Morphological Analyzer (BAMA)—up to version 3—was created by Tim Buckwalter in 2002 (Buckwalter, 2004; Habash, 2010) and it is one of the best known analyzers for the Arabic language. As Attia et al. (2011a) claims, it "is a de facto standard tool which is widely used in the Arabic NLP research community". The last available version is the 3.1. It is lexicon-based and presents a concatenative approach: Arabic morphology is treated as purely agglutinative, so the analysis algorithm focuses on how all the morphemes can be combined in a word. The phonological, morphological and orthographic alterations are simply codified in the lexicon: one same word may have more than one entry in the lexicon according to the number of lexemes its inflectional set of forms presents. The three lexicons are:

1. Lexicon of prefixes (1,328 entries)
2. Lexicon of suffixes (945 entries)
3. Lexicon of stems (79,318 entries representing 40,654 lemmas)

There are three tables of morphological compatibility which specify how the three lexicons combine among themselves:

1. Table of prefix-stem combinations (2,497 entries)
2. Table of stem-suffix combinations (1,632 entries)
3. Table of prefix-suffix combinations (1,180 entries)

Additionally, SAMA—as well as its older version BAMA—includes English glosses for all the lexical items. The analytical model can be described as a stem-based concatenative approach. It uses a transliteration that represents Arabic script.

Sawalha and Atwell (2008) carried out an evaluation experiment on accuracy of the Bama analyzer—the older version—as well as on two more morphological systems:

The Shereen Khoja Stemmer (Khoja & Garside, 1999) and a triliteral root extraction algorithm developed by Shalabi, Kanaan and Al-Sherhan (2003). As a whole, the three systems achieved about a 62% accuracy rate for Quranic texts and 70% for newspaper texts. However, the BAMA presented the worst results, with about 59% accuracy for Quranic text and 38% for newspapers. The low level of accuracy achieved in contemporary texts indicates that the lexicon of BAMA is more focused on classical vocabulary.

SAMA has been integrated and is used in several applications such as treebank annotation (Kulick et al., 2010) or lexicographic studies (Buckwalter and Parkinson, 2010).

5.3.5 MAGEAD (Habash et al.)

MAGEAD (Habash et al, 2005; Habash & Rambow, 2006; Altantawy et al., 2010; 2011) is a morphological analyzer and generator for MSA and the spoken dialects. It relates a lexeme and a set of linguistic features to a surface word form through a sequence of transformations. (Habash, 2001). MAGEAD follows a similar multi-tape representation as the one developed by Kiraz. The representation consists of five tiers:

Tier 1: pattern and affixational morphemes.
Tier 2: root.
Tier 3: vocalism.
Tier 4: phonological representation.
Tier 5: orthographic representation.

On the other hand, it describes three different types of morphemes:

1. Templatic morphemes, which can be of three types (root, pattern and vocalism) and together create a word stem.
2. Affixational morphemes, which are added to the stem depending on its "morphological behaviour class".
3. Non-templatic word stems, i.e. words which do not follow the innate Arabic system of templatic morphology.

Magead's verbal lexicon contains 8,960 lexemes. The transformations to build the surface form from the underlying form are rewrite rules based on Arabic phonological alterations and orthographic idiosyncrasies of the Arabic script. The transformational rules contain 69 Morphophonemic and phonological rules and 53 Orthographic rules.

The system is able to work with a partial lexicon or even without any lexicon. Lexemes may be hypothesized on the fly without having to make wild guesses. First it identifies the pattern and then it "guesses" the root. The system is bidirectional, it both analyzes and generates. Moreover, it does not only work with MSA, but also with Arabic dialects. It has been evaluated against the Penn Arabic Treebank (ATB)[64] and the Levantine Arabic Treebank (LATB).

[64] http://www.ldc.upenn.edu/Catalog/catalogEntry.jsp?catalogId=LDC2005T20

5.3.6 MADA+TOKAN and ALMORGEANA (Habash et. al)

MADA+TOKAN (Habash et al. 2009; Roth et al. 2007) is a toolkit which contains different NLP tools for processing Arabic language. The MADA system performs morphological analysis and disambiguation, including POS Tagging, Stemming, Lemmatization and diacritization. TOKAN, in turn, performs the tokenizing task. The package offers a whole range of tasks which can be used in many applications, such as Information Retrieval, Machine Translation or Named-Entity Recognition.

The system first provides a list of all possible analyses for each word of a given text. Words are marked with various grammatical features: part-of-speech, aspect, case, gender, mood, number, person, state, voice and clitics—at syntax level. The set of analyses of each word are ranked using weights so the analysis most suitable to the context has the higher weight. Weights are calculated by means of Support Vector Machines (SVMs) trained on the Penn Arabic Treebank (PATB). The system takes into account spelling variations and, additionally, uses information of n-gram statistics. The output of the system is a list of all possible analyses for each word, the first being the one with the higher score.

The morphological analysis provided by MADA uses the Almorgeana morphological analyzer. Additionally, it requires one of the following analyzers: BAMA, SAMA—this is the preferred tool of the MADA system—or Aramorph. MADA uses the chosen software to extract the information of prefix, stem and suffix tables, and incorporates it into a database which Almorgeana uses to do the actual morphological analysis.

ALMORGEANA is a lexeme-based morphological generator and analyzer for MSA. It is a version of the BAMA system adapted to morphological generation and analysis and focused on lexeme representation (Habash and Rambow, 2007). It uses the lexicons of BAMA. The database consists of six components; three morphological

databases (prefixes, stems and suffixes) and three compatibility tables for the databases (prefix-stem, stem-suffix and prefix-suffix). Its main features are:

a. Large lexical coverage
b. robustness in coverage of morphological and orthographic phenomena
c. Reversibility: it can be used as a generator or as an analyzer
d. Usability in a wide range of NLP applications
e. The generated forms are fully diacritized

In relation to its performance, MADA has over 96% accuracy on basic morphological choice—including tokenization but excluding case, mood, and nunation—and on lemmatization. On the other hand, it has over 86% accuracy in predicting full diacritization—including case and mood—(Habash et al., 2009).

5.3.7 A syllable-based account of Arabic morphology (Cahill)

Cahill (1990; 2007; 2010) describes a syllable-based formalism to deal with Arabic verbal and nominal morphology. The formalism follows the classification of the syllabic elements onset, peak and coda to describe the internal structure of Arabic stems. The morphophonological alterations caused by the presence of weak roots are treated in syllabic terms as well. Cahill claims that the model is elegant and has the advantage of not using different techniques to describe the various processes; besides it handles fully inflected forms. The approach is based on similar analyses proposed for describing the ablaut in Germanic languages.

In Cahill's approach, Arabic weak roots are completely regular morphologically. The morphological alternations encountered in these roots are determined by phonology. Therefore, she claims that those roots do not need any further specification in the lexicon.

Cahill states that the vast majority of Arabic verbs (at least the triliteral ones) are disyllabic. Therefore, she establishes that the verbal structure of Arabic is disyllabic by default. She handles triliteral and quadriliteral verbs separately. The relevance of this approach lies, basically, in that it is completely based on syllabic analysis. The model is still incomplete, but small testing has been carried out.

For the implementation, she uses the lexical representation language DATR which includes inheritance techniques. The DATR[65] implementation of the lexicon is based on the lexicon structure of PoliyLex (Cahill, 2010). The system uses SAMPA[66] for the phonemic representations.

5.3.8 ELIXIRFM (Smrž)

ElixirFM (Smrž, 2007a; 2007b; Smrž et al., 2008) is a project developed at the Institute of Formal and Applied Linguistics from Charles University and headed by Otakar Smrž. The project consisted of the implementation of a computational model of the morphological processes of MSA based on the Functional Morphology library for Haskell. His model takes an approach focused on the syntax-morphology interface. Smrž takes as a starting point the open-source Buckwalter lexicon, organizes it and enriches it with more information. Further, he enhances the ElixirFM morphological system with the Prague Arabic Dependency Treebank a multi-level linguistic annotated newswire texts of MSA. According to the Linguistic Data Consortium

[65] Moreno Sandoval and Goñi Menoyo (2002) presented a complete model of Spanish morphology implemented in DATR which, however, was not based on the syllable. This gives us an idea of the flexibility that DATR formalism has.

[66] The Speech Assessment Methods Phonetic Alphabet (SAMPA) is a computer-readable phonetic script build on ASCII characters and used for representing IPA.

(LDC)[67], the Prague Arabic Dependency Treebank 1.0 has 113,500 tokens of data annotated analytically and provided with the disambiguated morphological information.

BAMA's analyzes focus on a surface stem-based representation. ElixirFM, on the other hand, analyzes forms based on lexemes and features. The linguistic analysis follows a root-and-pattern approach, yet it presents an *ad hoc* categorization of Arabic patterns: lexemes are codified in the lexicon carrying its root and pattern information if appropriate. Thus, morphology is modelled in terms of abstract patterns, which contain the appropriate morphophonemic information, paradigms, grammatical categories, lexemes, and word classes. So the surface word forms to be analyzed are expressed as a combination of the derivation of a root and a pattern, and this is the internal representation of lexemes.

Smrž uses the transcription system ArabTEX to deal with the Arabic script and a Latin equivalent simultaneously. ElixirFm is written in the Haskell programming language, and the lexicon is supported by interfaces in Perl. There is also an online interface available. The program has various ways of accessing the information:

a. Analysis of a text, including tokenization
b. List of inflectional forms of a specific lexeme
c. Derivation of a word given its positional morphological tags or a description (in natural language)
d. Lookup for an entry by its English gloss

[67] http://www.ldc.upenn.edu/

5.3.9 Lexicon of Arabic verbs using FSTs (Neme)

Amid Neme (2011) generated a fully diacritized lexicon of inflected verbal forms using FSTs. FSTs are prototypically employed to describe languages using concatenative morphology procedures. However, the present work claims to demonstrate the effectiveness of FSTs to represent root-and-pattern morphologies. Stems are defined following the root-and-pattern approach. A prefix-stem-suffix representation is used for the FSTs.

The generation system includes three components: lexicon, rewrite rules, and morphotactics. The lexicon, in turn, includes a lexicon of prefixes, a lexicon of stems and a lexicon of suffixes. The lexicon of stems is precompiled based on a root-and-pattern representation. The morphotactics component deals with changes at morpheme boundaries such as deletion, epenthesis, and assimilation. The rewrite rules map the multiple lexical representations to a surface representation.

The lexicon of lemmas is taken as an input for the generation. This lexicon contains 15,400 verbal entries. Each entry includes the information of lemma, pattern and root. Patterns follow the traditional classification of Arabic grammar. Roots are redefined to mark explicitly the presence of special letters (*waw*, *ya* or *hamza*), giving rise to a total of 31 different classes. The 31 root classes are combined with 460 inflectional classes to generate all possible inflectional paradigms.

The generation was evaluated against 10,000 diacritized verb occurrences in the Nemlar corpus. It was also compared with Buckwalter resources.

5.3.10 AraComLex (Mohammed Attia)

AraComLex is a large-scale finite-state morphological analyzer toolkit for MSA developed principally by Mohammed Attia (Attia et al., 2011a; 2011b; Attia, 2005). It is based on the lemma as the basic lexical entry for the morphological analyzer. Attia notes that a stem-based system is more costly for it has to list all the stem variants of a form, whereas a lexeme-based system simply includes one entry for each lexical form and a set of generalized rules for handling the variations. He also rejects a root-based approach, as it is more complex and tends to cause overgeneration problems.

It relies on a lexical database specifically constructed for this toolkit. The lexical database was extracted from a contemporary MSA corpus of more than 1 billion words. The MADA toolkit was used to pre-annotate the corpus, and machine techniques as well as knowledge-based pattern matching were used to automatically acquire lexical knowledge. As a result, AraComLex is the only Arabic morphological analyzer which includes strictly MSA contemporary vocabulary. Attia et al. (2011b) estimate that about 25% of the lexical items included in SAMA are outdated.

The system is built on the lexc language, a phrase structure grammar. It codifies lexical entries along with all possible affixes and clitics. The input databases include a lexicon of lemmas (5,925 nominals and 1,529 verbs), a lexicon of patterns (456 for nominals and 34 for verbs), and the filtered root-lemma lookup lexical database from SAMA 3.1. There are 130 alteration rules to handle all alterations encountered in the lexicon. It was evaluated against a corpus and compared to SAMA. It showed a better performance in the analysis as it reduces considerably the ambiguity of SAMA. AraComLex toolkit is distributed under a GPLv3 license.

	Kiraz 1994a; 1996; 1999; 2000; 2001	Xerox (Beesley 1998b, 2001)	(Cavalli-Sforza, 2000)	SAMA 3.1 (Buckwalter, 2004)	Magead (Habash, 2005; 2010)	Cahill 2007, 2010	ElixirFM (Smrž and Bielicky, 2007)	Neme, 2011	AraComLex (Attia, 2011a; 2011b)
technology	multi-tape two-level morph., FSTs	two-level morph. into FSTs	MORPHE	-	FSTs	DATR PoliLex	-	FSTs	FSTs
programming language	-	perl, lexc, twolc	Lisp	perl	-	DATR	haskell and perl	-	lexc
linguistic model	prosodic morph. (root-and-pattern)	root-and-pattern	lexeme-based	concatenative stem-based	lexeme-based (root-and-pattern representation)	syllable-based	root-and-pattern (*ad hoc* pattern classification)	root-and-pattern defined in concatenative terms	lexeme-based
input lexicons	prosodic pattern, root, vocalism, affix	prefixes, roots (4,930), patterns (400) and suffixes	lexeme and features	prefixes (1,328), suffixes (945), stems (79,318), compatibility tables	root, pattern, vocalism, affixes	root, pattern and vowel inflections	-	15,400 verbs, 31 root classes, 460 inflectional classes	lemmas (7,454), patterns (490), root-lemma lookup database
grammatical coverage	-	overgeneration (but removed in a following step)	-	large-scale	large-scale (122 alteration rules)	partial	large-scale	-	large-scale (130 alteration rules)
transliteration	-	-	-	Buckwalter (2002)	-	SAMPA	Arabtex, Buckwalter	-	-
evaluation	-	-	partial	-	against ATB and LATB	small evaluation	-	against NEMLAR corpus	compared to SAMA on a corpus
availability	-	propriety software	-	open source	free for research	-	open source	-	open source

TABLE 14 Characteristics of the main morphological analyzers and models

The blank boxes indicate that the information was not found or is not applicable to the specific tool. All system work on fully vocalized word

Scope and objectives

Research on Arabic computational morphology has increased considerably in recent years. Indeed, research on Arabic morphology has always been extraordinarily prolific due to the complexity of the subject. However, despite the reasonable amount of computational models which have been proposed, the different approaches have not been completely explored and much more work is still needed. It is clear in the literature that any of the approaches appears to be better than the others from a general perspective.

Examining Arabic phonological traits and the Arabic verbal system as shown in the introduction, we determine that the most important characteristics of Arabic verbal grammar are the following:

a. Arabic shows a highly restrictive phonological system which enables a generalized formalization of syllabic structure. Khalil's prosodic theory to analyze poetry quantitatively is remarkable for its capacity to precisely describe syllabic structure.

b. There is just one inflectional system for all verbal patterns. Yet, the inflectional affixes of the paradigms are affected by syncretism.

c. The variation from underlying form and surface form is said to be formalizable, and thus treated by means of a set of phonological and orthographic alteration rules—several of the computational systems include a phonotactic and orthographic module to treat these alterations but, unfortunately, this set of rules is not readily available.

On the other hand, we believe that the classification of stems into root, prosodic template, affixation and vocalism—specially in the line of the adaptation of McCarthy's model as proposed by Kiraz (1994a; 1994b; 1996; 2000; 2001)—is interesting for they have proved to be successful as a descriptive tool—we have seen that the majority of Arabic processing systems follow a root-and-pattern model one way or another. The main problem of this approach, compared to others, is the architectural complexity it requires to develop an efficient and complete system. In this sense, we attempt to achieve a degree of abstraction which enables us to present a manageable and straightforward description of Arabic verbal morphology. Hence, our goals are as follows:

1. **Elegant and large-coverage formalization of Arabic verbal grammar** based on phonological principles and implemented in order to test the model: we attempt to present a complete and consistent computational model which accounts for Arabic verbal morphology—including morphophonological and orthographical alterations. The model adopts a lexical representation of the stem in prosodic template, root plus affixation and vocalism. The morphophonological operations are based on the syllabic structure and prosodic properties of Arabic verbs, and codified in the template. The accuracy of the linguistic description will be tested by means of a computational implementation.

2. **Creation of two lexical resources**—a lexicon of verbal lemmas and a lexicon of inflected verbal forms with linguistic feature specifications. A lexicon of verbal lemmas with root and inflectional specifications have been built to be used as the input of the computational implementation, which will be based on generation. The output of the generation system will be a broad coverage feature-based lexicon of inflected forms. We have seen that unification grammars need the word-forms of the language to include all morphological

information, as a previous step to describing syntactic structures. This means that morphology should be mainly treated in the lexicon.

3. **Evaluation of the lexicon of inflected forms returned by the computational model**. The lexicon of inflected forms returned by the generation system will be evaluated to ensure that the forms have been generated successfully.

4. **Development of an open-source morphological analyzer and generator which will have an online interface**: we have developed a morphological tool that is available under an open source license—there are not many open-source resources for Arabic, the most relevant at the moment being BAMA (Buckwalter, 2003) and AraComLex (Attia et al., 2011). The online interface allows queries to be sent to to the lexical databases.

Materials and methods

The computational system has been implemented in Python programming language (version 3.2). In recent years it has come to be one of the best options for developing applications in the field of natural language processing. Further, version 3 of Python fully supports Unicode, so it can directly handle Arabic script. In relation to orthography, we handle fully diacritized forms. Arabic uses diacritics to disambiguate words (Al Shamsi and Guessoum, 2006), and thus we keep this *tool* to create a lexicon as unambiguous as possible. The phonotactic constrain alterations, which cause a gap between the underlying—regularized—form and the surface form, were formalized using regular expressions. The code of the program is collected in Appendix E.

We have manually created a lexicon of Arabic verb lemmas which consists of 15,453 entries with unambiguous information of each verbal item. This database has two purposes: first and most obvious, the lexicon will be used as an input for the system of verbal generation. Second, and not less important, the lexicon will be used to extract quantitative data in order to describe our lexicon and, additionally, to support some of the hypotheses on Arabic morphological theory presented in the literature.

The lexicon was taken from a list of verbs included in the book "A dictionary of Arabic verb conjugation" by Antoine El-Dahdah (1991). We find it fundamental to take into account the lexicographical sources used by El-Dahdah to compose his lexicon, since we will base our analyzer on their reliability. The sources mentioned in the dictionary are widely known classical dictionaries:

محيط المحيط، قاموس مطوّل للغة العربية، المعلم بطرس البستاني، مكتبة لبنان، بيروت ١٩٧٧

المعجم المحيط، معجم اللغة العربية، شركة الإعلانات الشرقية ١٩٨٥

لسان العرب، ابن منظور، دار صادر، بيروت ١٩٦٨

القاموس المحيط، الفيروز آبادي، مؤسسة الرسالة، بيروت ١٩٨٦

"Muhit al-Muhit", Butros Bustani, Librairie du Liban, Beirut, 1977
"al-Mu'jam al-Muhit", Sharika al-i'lanat al-sharqiya, 1985
"Lisan al-'Arab" Ibn Manzur, Dar sader, Beirut, 1968 The Arab Tongue
"al-Qamus al-Muhit", Fairuzabadi, Mu'assasat al-Risala, Beirut, 1986 The comprehensive lexicon

As to a possible objection to our source material, let us return to the statement made in the introduction: Arabic lexicographic sources cover a heterogeneous linguistic entity; these reference books have the advantage of providing a large coverage corpus, but they evidently include mixed linguistic varieties, including not just MSA vocabulary but also Classical Arabic. Yet, it is a drawback common to all the lexicographical material available—except AraComLex (Attia et al., 2011), in which Attia avoided this drawback by using a contemporary Arabic corpus.

Apart from al-Dahdah and his sources, we relied on several works for consulting grammatical and lexical information and thus for developing the rules and the lexicon of the verbal generation system: Cowan (1958); Haywood and Nahmad (1962); Corriente (1970); المنجد في اللغة والاعلام (1973); Reig (1983); Corriente (1991); Wehr (1993); احمد مختار عمر et al. (1994); Cortés (1996); توفيق بن عمر بلطه جي (1997); نايف معروف (2000); Badawi et al. (2004); Ryding (2005); شهاب الدين أبو عمرو and يوسف بقاعي (2005); Hassanein (2006); Abu-Chacra (2007); Wright (2007); Mace (2007); الدحداح (2008); and the site قاموس الباحث العربي: عربي عربي http://www.baheth.info/.

For the evaluation task, we used the lexicon of inflected forms extracted from the ElixirFM Arabic morphological analyzer and generator. After normalizing the data to respect our conventions, we matched the word-forms in search of common entries and we established correct and incorrect forms from that subgroup.

Results

1 Linguistic formalization of the model

We believe that prosody has a fundamental role in Arabic morphology, and that the syllabic structure of Arabic verbal stems can be formalized in a reduced set of abstract structures. We rely this hypothesis on the fact that many of the morphological analyses of the Arabic system—both in the linguistic and in the computational fields—distinguish a CV-skeleton. Yet, perhaps the most relevant of the previous works is al-Khalil's quantitative prosodic theory, for it computes syllabic weight by means of a systematic and simple mathematical device based on orthography. Al-Khalil's counting procedure hints at the existence of an extremely regular system of syllabic structure in Arabic.

Taking this idea as a key point, the proposed model will essentially be based on the division of stems in three lexical items—a prosodic template, a root plus affixation amalgam, and a vocalization—and a formal device for merging these three items to build verbal stems.

Prosodic templates abstract the syllabic structure of the underlying representation of verbal stems measured in moraic units. We propose two types of templates which cover all the verbal derivations—the so-called verbal patterns—in the Arabic system. The basic difference between these two types of templates is the length of the penultimate syllable: on one type this syllable is heavy and on the other it is light. Hence, we are going to name the first type *H*, for *heavy*, and the second *L*, for *light*.

Both types distinguish a p-stem, an i-stem, and an m-stem[68], as each verb presents these three stems along its conjugation.

Template type	p-stem	i-stem	m-stem
L	FFVFWF	VFFFWF	FFFWF
H	FFVFFWF	VFFFFWF	FFFFWF

TABLE 15 Jabalín classification of templates

The symbol *F* represents consonants—including glides—either radicals or affixation, as well as vowel lengthening processes. The symbols *V* and *W*, in turn, represent vowels, the only difference between them being the position: V represents the first vowel and W the second. In terms of syllabic constituents, V and W correspond to the nucleus of the rime, and either compute as one mora. F elements correspond to all phonological segments covering onset position, coda position and the vowel lengthening feature in the complex nucleus. This means that within the rime, F segments are all segments which mark the syllabic weight as heavy, i.e. if an F is in the rime, the syllable computes two moras.

This formal representation uses just two different notations to represent the three basic syllabic structures of Arabic[69]:

light syllable	heavy syllable (closed)	heavy syllable (open)
CV	CVC	CVV
FV	FVF	FVF

TABLE 16 Equivalence of syllables and Jabalín notation

[68] Recall, we use p-stem, i-stem and m-stem to refer to perfective stem, imperfective stem and imperative stem respectively.

[69] Recall that *V* and *W* are identical within a syllable. In the formalism they simply distinguish the position of the syllable inside the stem template. In the following example we use *V*, but we could have represented it with *W* too.

We can see in both the L and H templates that there are F clusters, which could represent consonant clusters or sets of consonants plus vowel lengthening. These prohibited structures, if yhey occur after the insertion of the root plus affixation and vocalization, are removed in a following step in the form of a syllabic constraint rule—which is explained later in this section.

The two types L and H classify verbal patterns into two groups, depending on which of the templates the verbal pattern follows.

Patterns	Template
Iau, Iai, Iaa, Iuu, Iia, Iii, VII, VIII, IX	L
II, III, IV, V, VI, QII, X, XI, XII, XIII, XIV, XV, QI, QII, QIII, QIV	H

TABLE 17 Classification of patterns in templates L and H

F slots are filled with the root plus affixation material. V and W slots are filled with the elements of the vocalism item.

The root plus affixation amalgam consists of the set of consonants of the root and the derivational affixes arranged in order—with the only exception of the prefix ta- of patterns V, VI and QII. Even though it is a discontinuous construction, it is represented linearly. For instance, in the verb اِعْتَبَرَ *Aiçtabara*—meaning 'to consider'—from pattern VIII, we have a root √*çbr* and a derivational affix -*t*-. The root plus affixation amalgam is represented as *çtbr*, respecting the order of each of the consonants as it is arranged in the surface representation of the verb.

To construct the root plus affixation amalgam we have to determine the affixation material of each of the patterns and its exact position in relation to the root. As noted earlier, we do not represent the prefix ta- of patterns V, VI and QII at this level. Apart from this, the derivational process of vowel lengthening, which along with

consonants is represented as F in the template, will be indicated by the symbol *E* in the root plus affixation amalgam. Hence, in the verb حَاوَلَ *HaAwala*—meaning 'to try'—from pattern III, we extract a root √*Hwl* and an affix representing vowel lengthening 'E', comprising the amalgam *HEwl*.

In section 4.4.2.3 we listed the derivational material of each pattern. By carefully examining all the pattern derivations, we notice that the derivational material consists primarily of three processes:

(a) the gemination of consonants,
(b) the lengthening of vowels[70], and
(c) the insertion of consonantal affixes.

We classify these three processes in two operations: lengthening—in which we include vowel lengthening as well as consonant gemination, as they both lengthen the syllabic structure—and addition. In the following lines we list all the derivational material—except the ta- prefix—according to this binary categorization so as to fit in our formalization of the root plus affixation amalgam:

Lengthening

The last radical is duplicated	IX, XI, XIV, QIV
The second radical is duplicated[71]	II, V, XII
'E' is added between the first and the second radicals	III, XI
'E' is added at the end	XV

[70] Recall that long vowels are represented by a short vowel followed by ا *A*, و *w* or ي *y* depending on the vowel; thus, we have ـَا *aA* for long *a*, ـُو *uw* for long *u* and ـِي *iy* for long *i*.

[71] Due to orthographic reasons, this operation will be divided into two rules in the computational system, one affecting II and V, and the other XII.

Addition

أ *Á* is added at the beginning	IV, VI
ن *n* is added at the beginning	VII
ت *t* is added between first and second radicals	VIII
ست *st* is added at the beginning	X
وو *ww* is added between the second and the third radicals	XII
و *w* is added between the second and the third radicals	XIII
ن *n* is added between the second and the third radicals	XIV, XV, QIII
'E' is added between the second and the third radicals	XI

The last operation of addition should be classified as a lengthening operation according to our classification system. The reason for keeping this inconsistency in the classification is due to computational efficiency.[72]

The vocalization consists of a discontinuous string specifying the first vowel (V) and the second vowel (W) of the stem—the m-stem only includes the vowel W. P-stem and i-stem distinguish active from passive vocalization. Hence, the vocalization discriminates five stems: p-stem active, p-stem passive, i-stem active, i-stem passive and m-stem—active by default. In table 10 we presented the vocalization of each of the

[72] In the computational model each verb lemma has a code associated with it—it is described in the next section. The code indicates the mophological characteristics of that lemma in a manner that the affixes of the derived patterns are specified in the code. We reserve just 1 digit for codifying lengthening operations and another digit for codifying addition opetarions. This means we do not expect to find a verb with two operations of lengthening or addition, yet it happens in the case of pattern XI, in which two lengthening operations occur. This is the only exception to our assumption and it is a classical pattern—no longer in use in the system—restricted to a small set of verbs. For this reason, we preferred to keep the codification system as it was originally designed and include one of the lengthening operatons of pattern XI as an operation of addition.

patterns in active voice—passive voice was said to be invariantly *ui* for p-stem and *ua* for i-stem. Below, this same table is included again.

Pattern		Vocalization		
		p-stem	i-stem	m-stem
triliteral	Iau	aa	au	u
	Iai	aa	ai	i
	Iaa	aa	aa	a
	Iuu	au	au	u
	Iia	ai	aa	a
	Iii	ai	ai	i
	II	aa	uai	ai
	III	aa	uai	ai
	IV	aa	ui	i
	V	aa	aaa	aa
	VI	aa	aaa	aa
	VII	aa	aai	ai
	VIII	aa	aai	ai
	IX	aa	aa	a
	X	aa	aai	ai
	XI	aa	aai	ai
	XII	aa	aai	ai
	XIII	aa	aai	ai
	XIV	aa	aai	ai
	XV	aa	aai	ai
quadriliteral	I	aa	uai	ai
	II	aa	aaa	aa
	QIII	aa	aai	ai
	QIV	aa	aai	ai

We associate all first vowels to V, except in m-stems. Likewise, the last vowel is associated to W. In cases where there is a middle vowel—the first vowel in m-stems with two vowels—we do not consider this middle vowel at this stage; it will be inserted in the stem at a subsequent stage, by means of a syllabic constraint rule—the one we referred to when noting that some templates have the three Fs.

Therefore, in our formalization, we associate V and W values to different stems and voices. The vocalism tier shows some default values which affect all the patterns.

In p-stem active V is always ◌́ *a*
In p-stem passive V is always ◌̍ *u* and W is always ◌̣ *i*
In i-stem passive V is always ◌̍ *u* and W is always ◌́ *a*

In the rest of the cases, the vocalization depends on the specific pattern the verb has.

In p-stem active W can be ◌́ *a*, ◌̍ *u*, or ◌̣ *i*
In i-stem active V can be ◌́ *a* or ◌̍ *u*
In i-stem active and m-stem W can be ◌̣ *i*, ◌̍ *u* or ◌́ *a*

STEM	TEMPLATE POSITION	VOWEL	PATTERNS
p-stem active	V	a	all
p-stem passive	V	u	all
	W	i	all
i-stem passive	V	i	all
	W	a	all
p-stem active	W	a	Iau, Iai, Iaa, II, III, IV, V, VI, VII, VIII, IX, X, XI, XII, XIII, XIV, XV, QI, QII, QIII, QIV
		u	Iuu
		i	Iia, Iii
i-stem active	V	a	Iau, Iai, Iaa, Iuu, Iia, Iii, V, VI, VII, VIII, IX, X, XI, XII, XIII, XIV, XV, QII, QIII, QIV
		u	II, III, IV, QI
i-stem active and m-stem	W	i	Iai, Iii, II, III, IV, VII, VIII, IX, X, XI, XII, XIII, XIV, XV, QI, QIII, QIV
		u	Iau, Iuu
		a	Iaa, Iia, V, VI, QII

TABLE 18 Classification of vocalism in Jabalín

The root plus affixation and the vocalism are inserted into the prosodic template through a simple procedure. This is the algorithm for inserting the root+affixation into the template:

Each segment of the root plus affixation amalgam substitutes each of the Fs in the template from the end to the beginning. In the same way, the vocalism substitutes the corresponding V and W elements of the template. When the insertion of all segments is completed, if there are any remaining Fs in the template, they are simply removed. The resulting construction is the verbal stem.

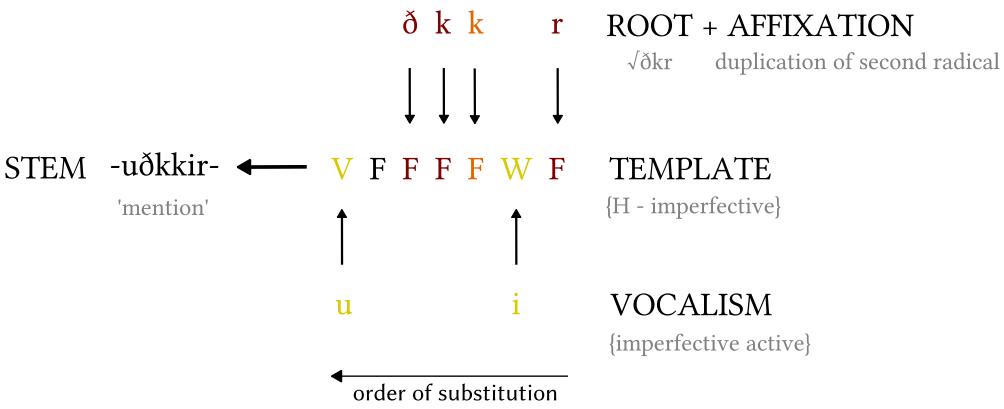


FIGURE 17 Algorithm for inserting root+affixation in prosodic template
The stem from the example, -uðkkir- still needs to pass through another process to be a well-formed stem.

At this point the prefix ta- is added to verbal stems of patterns V, VI and QII. In p-stem and i-stem it is directly prefixed to the stem unit. In i-stem, it is located after the vowel V.

	p-stem	i-stem	m-stem
Template	ta+FVFFWF	V+ta+FFFWF	ta+FFFWF
eg. Pattern II	ta+faççal	a+ta+faççal	ta+faççal

TABLE 19 Insertion of prefix ta- in templates

In the following step, stems suffer some phonological alterations, before being inserted in their inflectional paradigm to create conjugated forms in underlying representation. Subsequently, phonotactic operations and an orthographic

normalization are applied to map the underlying forms with their surface representation. The phonological alterations in the stem domain are as follows:

a. The derivational prefix أ *Á* of all patterns IV is lost in i-stem and m-stem. This can be explained because the glottal stop prefix is relaxed in this context and ultimately removed:

type of stem	***regularized*** stem	correct stem	gloss
active i-stem	أُنزِل uÁnzil	نُزِل unzil	'reveal'

b. Assimilation of vowel *a* into *u* in the detransitive prefix *–ta* (patterns V, VI and QII) in passive p-stems.

type of stem	***regularized*** stem	correct stem	gloss
passive p-stem	تَنوِل tanuwwil	تُنوِل tunuwwil[73]	'was taken'

c. Avoidance of consonant clusters or consonants plus vowel lengthening mark clusters: all stems containing these clusters insert a vowel *a* to break the phonotactic constraint which disallows these constructions.

In L-template			
type of stem	***regularized*** **stem**	**correct stem**	**gloss**
active i-stem	عتبِر açtbir	عتَبِر açtabir	'consider'

[73] In this example we find the sequence *wi*, which tends to be prohibited in most contexts.

In H-template			
type of stem	***regularized* stem**	**correct stem**	**gloss**
active i-stem	ـَستحقِق astHqiq	ـَستَحقِق astaHqiq	'deserve'

In the second example, the stem resulting from the insertion of *a* in the cluster still needs to be changed by a phonotactic constraints rule, so that the stem has its correct shape: ـَستَحِقّ astaHiq~.

d. Assimilation of segmental features in pattern VIII. In section Introduction-3.4 we noted that derivational affix -t- of pattern VIII may assimilate with the first consonant of the verbal root. We describe these alterations in four types of changes[74]:

1. semiconsonants /w/ and /y/ are weakened and completely assimilated by /t/.
2. alveolar segments /þ/, /ð/, and /Z/ assimilate the /t/ affix.
3. voicing of /t/ to /d/ in contact with alveolar voiced segments /z/ and /d/.
4. /t/ is emphasized when preceded by an emphatic /D/, /S/ and /T/.

The inflectional paradigm is widely affected by syncretism. For this reason, we preferred not to segment any of the morphemes according to the inflectional features person, number and gender, but to keep them as a whole. However, in the imperfective paradigm we distinguish a set of inflectional affixes marking the imperfective from the terminations indicating the mood—indicative, subjunctive and jussive. For example, if we inflect in second person dual (2DN) the active i-stem *açtabir* 'consider', the circumfix *t_aA*—indicated in the columns 'All'—is added to the

[74] The formalization of the rules in section 2.2.5 present some variations due to orthographic reasons.

stem in the first place: *t-açtabir-aA*. Then, if we inflect in indicative mood, we have to add the suffix *-n*i. The resulting inflected form is *taçtabiraAni* ‘you both consider'.

Inflection	Perfective	Imperfective				Imperative
		All	Indicative	Subjunctive	Jussive	
1SN	-تُ -tu	أ- Á-	-ُ -u	-َ -a	-	*None*
1PN	-نَا -aAni	ن- -na	-ُ -u	-َ -a	-	*None*
2SM	-تَ -ta	ت- t-	-ُ -u	-َ -a	-	-
2SF	-تِ -ti	ت-ِي t-iy	-نَ -na	-	-	-ِي -iy
2DN	-تُمَا -tumaA	ت-َا t-aA	-نِ -ni	-	-	-َا -aA
2PM	-تُمْ -tum	ت-ُو t-uw	-نَ -na	-ا -A	-ا -A	-ُوا -uw
2PF	-تُنَّ -tun~a	ت-نَ t-na	-	-	-	-نَ -na
3SM	-َ -a	ي- y-	-ُ -u	-َ -a	-	*None*
3SF	-َت -at	ت- t-	-ُ -u	-َ -a	-	*None*
3DM	-َا -aA	ي-َا y-aA	-نِ -ni	-	-	*None*
3DF	-َتَا -ataA	ت-َا t-aA	-نِ -ni	-	-	*None*
3PM	-ُوا -uwA	ي-ُو y-uw	-نَ -na	-ا -A	-ا -A	*None*
3PF	-نَ -na	ي-نَ y-na	-	-	-	*None*

TABLE 20 Inflectional paradigm

After having built the stem and inserted the inflectional affixes in it, we have an underlying representation of the verbal form. We already mentioned that some phonological and orthographic constraints are applied to the underlying representation to get the targeted surface form. Orthographically, we must include the sukun symbol in the forms and adapt the hamza to its correct form according to the context[75]. The phonological alterations, of purely linguistic nature, are divided in three groups

1. The alterations caused by the semiconsonants w and y—including the loss of these segments of the alteration of the adjacent segments.
2. The gemination of consonants in specified contexts (operations of *tashdid*).
3. The breaking of initial consonant clusters—described in Introduction-3.2.

The rules developed for the treatment of both the orthographic and the phonological alterations are explained in section 2.2.7 and the full list can be found in Appendix E, inside the python code.

[75] In Introduction-2.2 we explained reasons for the ambiguous representations of the letter hamza in the forms ء *c*, أ *Á*, إ *À*, ئ *ý*, ؤ *ú* and آ *Ã*.

2 A computationally motivated model of verbal morphology

The core idea behind the system of verbal generation is that we classified Arabic verbal system in only two conjugational classes—considering both derivational and inflectional processes as part of the conjugation—and irregular categories are simply non-existent. Therefore, we designed a generation model which treats all verbs under a single frame, i.e. as if they all were *regular*—understood in the traditional sense, verbs without weak, *hamza* or doubled letters in their root. We have seen that phonotactic constraints are responsible for the surface variations and *abnormalities* found in some forms, and consequently, these variations are treated after the *regularized* conjugational phase, in the last stage of the generation system.

The verbal generation system is developed under the Jabalín project. Until now, Jabalín includes the verbal generation system, an online interface for analyzing and conjugating Arabic verbs, a transliteration scheme for the Arabic alphabet, the evaluation carried out on the generation system output lexicon and some quantitative data extracted from the lexicons. The online interface is described in Results-4. The conventions used for the transliteration are explained in Appendix A, and the quantitative data is included in Appendix B.

That said, the generation process makes use of two components: a lexicon of verb lemmas and a generation system. The generation system is the core part of the process; it comprises a set of different modules which treat the different phenomena of Arabic morphology sequentially. This generation system takes the lexicon of lemmas as input and returns a lexicon of inflected forms together with relevant linguistic information.

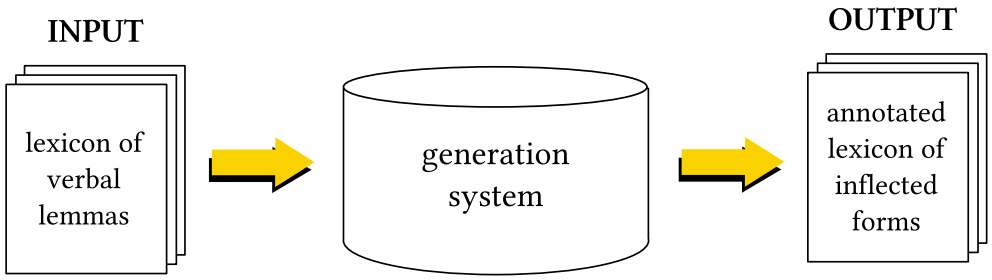


FIGURE 18 System of verbal generation

2.1 Lexicon of verbal lemmas

The lexicon of verb lemmas is a database of Arabic verbs. Each entry corresponds to a single verb and includes three pieces of information: lemma, root and code. These three elements act as an unambiguous identifier for each verb. Examples of lexicon entries:

lemma	root	code
استقبل	قبل	04H0000
كان	كون	00L0001
اظّنّ	ظنن	02L0000

We pointed out in section 3 that the lexicon was taken from a list of verbs included in the book "A dictionary of Arabic verb conjugation" by Antoine El-Dahdah (1991). Various classical dictionaries were used as lexicographical sources—they are listed in section 3.

This reference work developed by Al-Dahdah is a compendium of all possible conjugational squemes of Arabic verbs. El-Dahdah makes a classification of Arabic verbal conjugation types according to the derived patterns and the nature of root

letters—if they contain weak letters, hamza or identical segments. The classification follows a computational-like organization of the different verbal types. Therefore, this classification was taken as a starting point for the generation system.

The classification is specified by a code, which is attached to each verb. This code consists of three digits:

1. the first digit indicates if the verbal root is triliteral or quadriliteral and if the verb is simple or derived;
2. the second digit indicates the verbal pattern;
3. the third digit indicates the nature of the verb, i.e. if it contains weak letters, hamza or if the second and third radical are identical.

فعل	وزن	سالم	مضاعف	مهموز الفاء	مهموز العين	مهموز اللام	معتلّ الفاء	معتلّ العين	معتلّ اللام	مفرزق	مقرون
مجرّد ثلاثي	فَعَلَ_	١١٠	١١١	١١٢		١١٤	١١٥	١١٦	١١٧		
	فَعَلِ_	١٢٠	١٢١	١٢٢	١٢٣		١٢٥	١٢٦	١٢٧	١٢٨	١٢٩
	فَعَلَ_	١٣٠	١٣١	١٣٢	١٣٣	١٣٤	١٣٥	١٣٦	١٣٧		
	فَعُلَ_	١٤٠	١٤١	١٤٢	١٤٣	١٤٤	١٤٥	١٤٦	١٤٧		
	فَعِلَ_	١٥٠	١٥١	١٥٢	١٥٣	١٥٤	١٥٥	١٥٦	١٥٧	١٥٨	١٥٩
	فَعِلِ_	١٦٠					١٦٥			١٦٨	
مزيد ثلاثي	فَعَّلَ	٢٠٠	٢٠١	٢٠٢	٢٠٣	٢٠٤	٢٠٥	٢٠٦	٢٠٧	٢٠٨	٢٠٩
	فَاعَلَ	٢١٠	٢١١	٢١٢	٢١٣	٢١٤	٢١٥	٢١٦	٢١٧	٢١٨	٢١٩
	أَفْعَلَ	٢٢٠	٢٢١	٢٢٢	٢٢٣	٢٢٤	٢٢٥	٢٢٦	٢٢٧	٢٢٨	٢٢٩
	تَفَعَّلَ	٢٣٠	٢٣١	٢٣٢	٢٣٣	٢٣٤	٢٣٥	٢٣٦	٢٣٧	٢٣٨	٢٣٩
	تَفَاعَلَ	٢٤٠	٢٤١	٢٤٢	٢٤٣	٢٤٤	٢٤٥	٢٤٦	٢٤٧	٢٤٨	٢٤٩
	إِنْفَعَلَ	٢٥٠	٢٥١	٢٥٢		٢٥٤	٢٥٥	٢٥٦	٢٥٧		٢٥٩
	إِفْتَعَلَ	٢٦٠	٢٦١	٢٦٢	٢٦٣	٢٦٤	٢٦٥	٢٦٦	٢٦٧	٢٦٨	٢٦٩
	إِفْعَلَّ	٢٧٠						٢٧٦	٢٧٧		
	إِسْتَفْعَلَ	٢٨٠	٢٨١	٢٨٢	٢٨٣	٢٨٤	٢٨٥	٢٨٦	٢٨٧	٢٨٨	٢٨٩
	إِفْعَوْعَل...	٢٩٠	٢٩١	٢٩٢			٢٩٥	٢٩٦	٢٩٧		٢٩٩
مجرّد رباعي	فَعْلَبَ	٣١٠	٣١١		٣١٣	٣١٤	٣١٥	٣١٦	٣١٧	٣١٨	
مزيد رباعي	تَفَعْلَبَ	٤١٠	٤١١		٤١٣	٤١٤	٤١٥	٤١٦	٤١٧	٤١٨	
	إِفْعَنْلَبَ	٤٢٠				٤٢٤		٤٢٦	٤٢٧		
	إِفْعَلَبَّ	٤٣٠				٤٣٤		٤٣٦			

TABLE 21 Classification of the system of codes used by Al-Dahdah (Arabic)

Verb	Pattern	Reg	R2=R3	R1ç	R2ç	R3ç	R1wy	R2wy	R3wy	R1&R2wy	R1&R3wy
Triliteral Simple	Iau	110	111	112		114	115	116	117		
	Iai	120	121	122	123		125	126	127	128	129
	Iaa	130	131	132	133	134	135	136	137		
	Iuu	140	141	142	143	144	145	146	147		
	Iia	150	151	152	153	154	155	156	157	158	159
	Iii	160					165			168	
Triliteral Derived	II	200	201	202	203	204	205	206	207	208	209
	III	210	211	212	213	214	215	216	217	218	219
	IV	220	221	222	223	224	225	226	227	228	229
	V	230	231	232	233	234	235	236	237	238	239
	VI	240	241	242	243	244	245	246	247	248	249
	VII	250	251	252		254	255	256	257		259
	VIII	260	261	262	263	264	265	266	267	268	269
	IX	270						276	277		
	X	280	281	282	283	284	285	286	287	288	289
	XI-XV	290	291	292			295	296	297		279
Quadriliteral Simple	I	310	311		313	314	315	316	317	318	
Quadriliteral Derived	II	410	411		413	414	415	416	417	418	
	III	420				424		426	427		
	IV	430				434		436			

TABLE 22 Classification of the system of codes used by Al-Dahdah (English)

The information in the table follows a left to right direction to be consistent with the English equivalent below. Reg: regular; R1ç *hamzated* first radical; R2=R3 means that second and third radicals are identical; R2ç hamzated second radical; R3ç hamzated third radical; R1wy weak first radical; R2wy weak second radical; R3wy weak third radical

Results

In the following table, we can see some examples of verb-code pairs, as presented in al-Dahdah, and the information associated:

verb lemma	code	information associated
وَفَى wafaY	128	Triliteral root, simple verb, pattern Iai, the first and third radicals are waw or ya
تَدَرَّبَأَ tadarbaÁa	414	Quadriliteral root, derived verb, pattern V, the third radical is hamza
تَحَدَّدَ taHad~ada	231	Triliteral root, derived verb, pattern V, the second and the third radicals are identical

TABLE 23 Example of al-Dahdah system of verb codification

In relation with our approach to verbal morphology, the classification proposed by El-Dahdah presents some drawbacks[76]:

1. Shared code for group of patterns: the derived forms from the triliteral verb XI-XV share the same code, causing unmanageable computational ambiguity.

2. Traditional system of conjugational classes: our formalization does not follow the traditional classifications of verbs in patterns, but, conversely, organizes Arabic verbs into just two conjugational classes. Al-Dahdah's code system presents a syncretic marking of the morphological material: either the vocalism morpheme or the derivational affixes are codified in a confusing and mingled way. For instance, the prefix 'ta-' of forms V, VI and quadriliteral II is marked in the second digit of the code as 5, 6 and 2 respectively.

[76] These problems in fact reflect that this classification has a pedagogical aim.

3. Classification of types of roots in regular and irregular classes: as we claimed that there are no irregular classes in Arabic verbal system, this classification is useless for our model[77].

These three points justify the need for a new system of code classification for Arabic verbs, consistent with our approach. The new system of codes must be:

1. morphologically unambiguous;
2. non-redundant;
3. and consistent with our generation model, i.e. the code is going to serve as a path along the modules for applying the exact operations at each stage of the generation system.

2.1.1 Description of the Code

The code consists of six digits and one letter, all presenting positional value. Each of the code digits is used for one of the processes of the generation system, in a sequential way, thus each one contains one kind of relevant morphological information—that must be given lexically. Below, the general content of all position digits is explained. The generation system will be explained later in detail.

1	2	3	4	5	6	7
Internal derivation	Internal derivation	Prosodic Template	External derivation	Vocalization Perf V2	Vocalization Imperf V1	Vocalization Imperf V1

[77] Indeed, if we wanted to treat phonological alterations as different morphological classes, the code presents an additional problem: verbs with two different types of 'irregularities' in its root—weak and hamzated, weak and doubled, doubled and hamzated—are not codified.

Position 1 and 2: Internal derivation

These codes mark the derivational affixes or processes the verb presents—except for the prefix ta- of patterns V, VI and QII, which is coded as External Derivation in position 4. For example, the lemma اِنْفَعَلَ Ain·façal of pattern VII will mark the presence of a prefix n- in its code. Position 1 codifies derivation based on processes of consonantal gemination—affecting radicals—and vowel lengthening; whereas position 2 codifies the addition of affixal material. The same verb may have two derivational processes; this is why we use two positions in the code for marking this information.

Position 3: Prosodic template

As we saw before, the highly restrictive phonotactic system of Arabic makes the syllabic structure of each verb type predictable. This led us to distinguish to templates, named L and H. Position 3 marks the set of prosodic templates, L or H, the verb follows.

Position 4: External derivation

This code marks only the presence or the absence of the derivational prefix *ta-*.

Positions 5, 6, 7 Vocalization

These positions mark the vowels which cannot be inferred grammatically, and must be known *a priori*, i.e. they are lexically or semantically determined. Position 5 indicates the second vowel (W) of active p-stem; position 6, the first vowel (V) of active i-stem; and position 7, the second vowel (W) of active i-stem. The first vowel (V) of active p-stem and both the first and second vowels passive constructions have default values, so this information does not need to be included in the code.

2.1.2 Generation of roots

The root was generated semiautomatically; it was extracted from the lemma string with the help of the information specified in al-Dahdah's code. Additionally, an extensive manual revision was carried out to fix exceptional cases.

For instance, lemmas of the pattern VIII—without vowels—are expected to be composed of an orthographic *alif* (*A*), plus a derivational infix -*t*- and a triliteral root. The derivational infix is located between the first and the second radicals. If we remove the alif and the infix -t- letters, the remaining material will be the root.

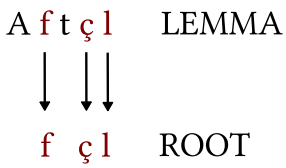


FIGURE 19 Extraction of root

Extraction of root by the procedure described above:

verb lemma	root
ارتبط ArtbT	ربط rbT
اتّجه Attji	تجه* *tjh

The root of the first example is correct. However, in the second one the root is wrong. The lemma of the second example has suffered an assimilation process between the root and the affix -t-; the root is in fact wjh وجه. This and other exceptional cases have been manually corrected.

2.2 Description of the Verbal Generation System

The generation System consists of a cascade of processes applied sequentially which take each root of a specified verb from the lexicon as an input, and returns a list of inflected forms together with a defined tagset of linguistic features. From a general perspective, the processes are divided into seven modules, which control the progress of the different generation stages, from root to stems, and finally to inflected forms. The seven modules are called:

1. Internal Derivation
2. Prosodic Template
3. External Derivation
4. Vocalization
5. Phonotactic preprocessing Module (Stem adjustment)
6. Inflection
7. Phonotactic Constraints and Orthographic Normalization

More specifically, the system takes each entry of the lexicon, consisting of a lemma, a root and a code, and inserts it into the sequence of modules. The root string is the element that will be extensively modified along the modules to produce the list of inflected forms. The code, on the other hand, indicates the path the root is going to follow through the whole generation process—so it acts as a tracking code for the generation process. The lemma, as specified in the input lexicon, is saved for the final display and, additionally, it may be used in a few rules from the last module, in the phonotactic constrain rules. The feature information is collected and stored throughout the whole process and attached to the final output.

The output of the Vocalization Module is a set of stems associated to that verbal root. The generation modules consist of the following procedures: first the root is merged with the derivational affixes and processes of the verb—none if the verb is simple—in

the Internal Derivation Module; the resulting combination is inserted in its corresponding prosodic templates; the External Derivation module adds another derivational affix in case the verb marks it; then the vocalic material is inserted as well in the string, creating six stems depending on voice and aspect; the six stems pass through a set of phonotactic preprocessing rules. After that, they are inserted in the different conjugation paradigms, in the Inflection Module. The last process is to pass the resulting forms though a sequential set of Phonotactic constraints and Orthographic rewrite rules, so impossible contexts are mended—in the figure below, we directly refer to this phase as *Phonotactics*. The final result is a lexicon of correct inflected forms. A set of features, collected throughout the process, is associated with each of these forms.

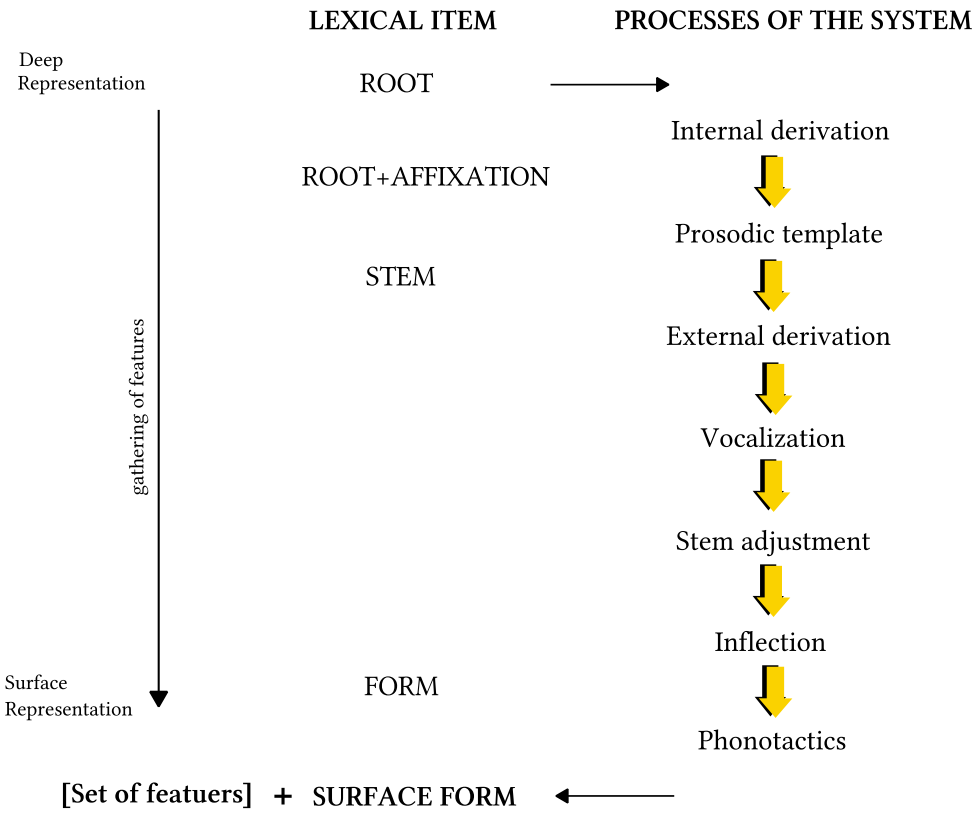


FIGURE 20 Modules of the generation system

The aim of the generation system is to have a complete lexicon of inflected verbal forms. The generation processes aim to reflect a theoretical description of the derivational, inflectional and phonological processes of Arabic, in the form of clear and detailed rules—as the computational treatment of natural language is *filtered* by orthography, orthographical processes are included as well. The majority of these rules are formalized by means of regular expressions. The convention followed for representing rules in the form of regular expressions is explained in detail in section 2.2.5.

2.2.1 Internal Derivation Module

This module receives a root as an input. The root is merged with the derivational material. The combination of the root plus the derivation affixes is the output of the module. Positions 1 and 2 of the code indicate the operations needed to join the root with the derivational affixes. Position 1 indicates the derivational affixes classified as operations of lengthening; position 2 indicates operations of addition [78]. The lengthening of vowels—which is a derivational process—is marked by the addition of the symbol 'E' in the specified position, which will be converted into a definite vowel in a future normalization processing.

As stated before, the generation path is controlled by the system of codes. The Internal Derivation operations are the following:

[78] Exceptionally, there is one operation of addition that is in fact an operation of lengthening. In note 71, we explained the reason for maintaining this inconsistency.

If **position 1** of code	**0**	there are no changes
	1	the last character is duplicated
	2	‘ً’ is added between the second and the third characters
	3	‘E’ is added between first and second characters
	4	‘E’ is added at the end
	5	the second character is duplicated

If **position 2** of code	**0**	there are no changes
	1	‘ن’ (‘n’) is added at the beginning
	2	‘ت’ (‘t’) is added between first and second characters
	3	‘ا’ (‘Á’) is added at the beginning
	4	‘ست’ (‘st’) is added at the beginning
	5	‘E’ is added between second and third characters
	6	‘و’ (‘w’) is added between second and third characters
	7	‘وو’ (‘ww’) is added between second and third characters
	8	‘ن’ (‘n’) is added between second and third characters

2.2.2 Prosodic Template Module

Each root plus affixation string will pass through three different templates: a p-stem template, an i-stem template and an m-stem template. There are only two possible sets of templates, L and H. Patterns Iau, Iai, Iaa, Iuu, Iia, Iii, VII, VIII and IX follow the L template set. The rest of the verbs follow type H. Each prosodic template indicates

the positions of all the root plus affixation consonants with the letter F. The vowels of the stem are marked as V, first vowel, and W, second vowel.

Value Position 2	p-stem	i-stem	m-stem
L	FFVFWF	V-FFFWF	FFFWF
H	FFVFFWF	V-FFFFWF	FFFFWF

The module takes as input a string consisting of a root plus affixation and returns three stems—still without specifying the vocalic melody—as output: perfective stem, imperfective stem and imperative stem.

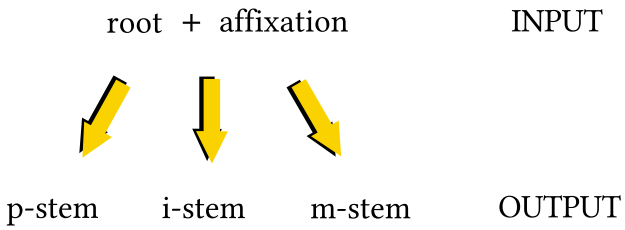


FIGURE 21 Prosodic template Module

Each character of the root plus affixation string replaces the Fs of the template starting from the end to the beginning. If there are some F symbols left after the replacement process, they are removed from the resulting string.

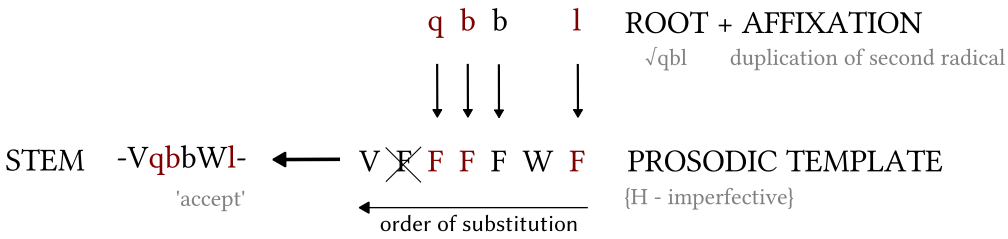


FIGURE 22 Insertion of root+affixation into the template

2.2.3 External Derivation Module

The External Derivation Module adds the derivational prefix ‘–تَ’ ‘ta-’ to the stems of the forms V and VI of the triliteral verb, and II of the quadriliteral. In the i-stem template, it is added between the first vowel and the first consonant of the form.

Value Position 3

0	None
1	is added at the beginning of the metrical unit of the stem in p-stem ‘تَ’ (‘ta’) is added at the beginning in i-stem ‘تَ’ (‘ta’) is added between the first and second characters in m-stem ‘تَ’ (‘ta’) is added at the beginning

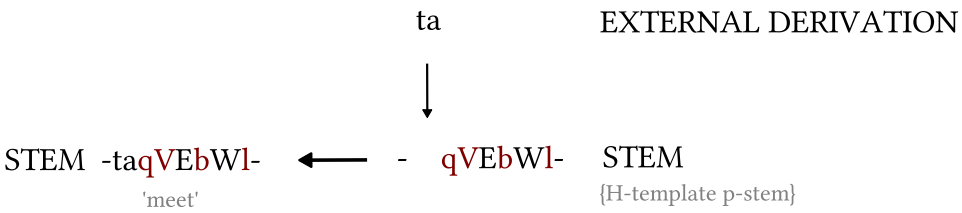


FIGURE 23 Example of ta- insertion in p-stem

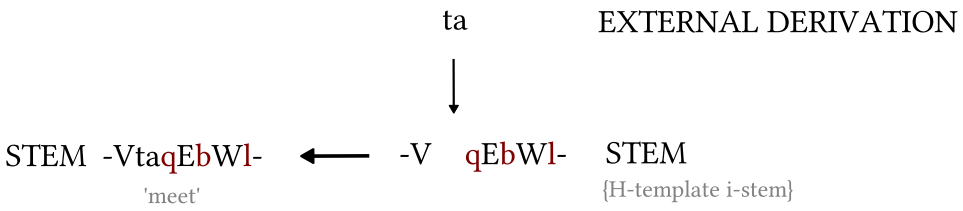


FIGURE 24 Example of ta- insertion in i-stem

2.2.4 Vocalization Module

The vocalization string distinguishes active and passive voice. Additionally, in the active voice the different patterns present different vocalic melodies. The vocalization module must indicate the vocalic material for all these grammatical conditions. First, default cases are specified for both first vowel (V) and second vowel (W) positions: the first vowel of active perfective is always '*a*'. The passive voice follows the melody '*u-i*' for perfective and '*u-a*' for imperfective, without exception.

Default values:

Active p-stem	V = 'a'
First vowel Perfective Passive	V = 'u'
Second vowel Perfective Passive	W = 'i'
First vowel Imperfective Passive	V = 'u'
Second vowel Imperfective Passive	W = 'a'

Position 6 indicates the **second vowel** of **p-stem active**

0	W = 'a'
1	W = 'u'
2	W = 'i'

Position 7 indicates the **first vowel** of **i-stem active**

0	V = ‘a’
1	V = ‘u’

Position 8 indicates the **second vowel** of **i-stem active and m-stem**

0	W = ‘i’
1	W = ‘u’
2	W = ‘a’

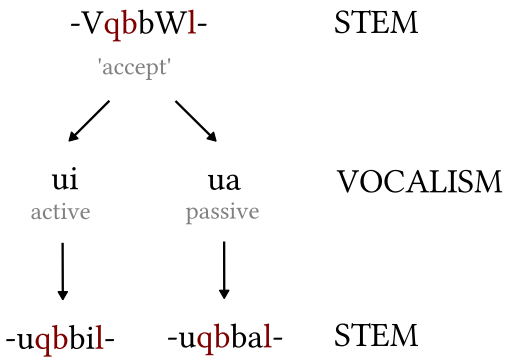
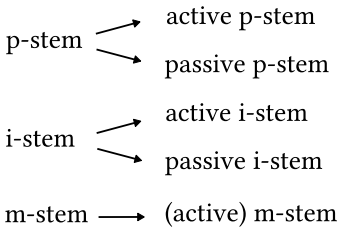


FIGURE 25 Insertion of the vocalism morpheme into the stem

The module receives three stems without vocalization from the previous module: a p-stem, an i-stem and an m-stem. It returns five stems: active p-stem, passive p-stem, active i-stem, passive i-stem and m-stem.



2.2.5 Stem adjustment Module

At this stage, a phonotactic preprocessing is applied to the outcome of the previous modules i.e. the generated five stems. There are 4 general rules for stem adjustment and 5 rules dealing with the assimilation processes of pattern VIII.

The rules of this module and the rules contained in the module of phonotactic constraints and orthographic normalization have identification codes so that they can be easily referred to. Rules are divided into groups. A letter is assigned to each group by alphabetic order and, within a group, each rule is assigned a number in ascending order. Hence, the rules in the first group will be listed as A1, A2, A3, etc; the rules in the second group will be B1, B2, B3, B4, etc; and so on. This modules includes the rules of groups A and group B. We will see in section 2.2.7 that the modules of phonotactic constraints and orthographic normalization include the groups of rules from group C to group G.

In the case of group B and all the rules included in the module of phonotactic constraints and orthographic normalization—with the only exceptions of C1 and E1—the alteration rules adopt the formal expression of rewrite rules. Rewrite rules are represented using the following convention:

a -> b / _c

If you find a *in the word-form, and if a is followed by c, then change a to b; where a is the pattern we are looking for, b in the replacement for the pattern, and c is the surrounding context; and the underscore indicates the position of a in relation to c.*

For example, the rule { ch ->hs / mat_ } means that the string 'ch' must be replaced by the string 'hs' only if it is preceded by the string 'mat'. In the example (7) below, the input is changed by the rule and returned a different string. However, in example (8) no replacement is applied, as the rule does not match any *pattern* in the input.

input	rule	output
match	ch ->hs / mat_	maths
catch	ch ->hs / mat_	catch

Some special characters used in regular expression are as well used for the representation of the rewrite rules:

1. The interrogative symbol '?' indicates one or zero occurrences of the previous character.
2. Square brackets '[ ]' or parenthesis plus a vertical bar '(|)' represent optionality among the characters contained inside.
3. The caret '^' means beginning of word.
4. the dollar '$' means end of word.
5. A caret inside square brackets '[^]' indicates that the string is matched only if the characters inside the square brackets are not present.
6. A dot '.' indicates any character.

For the sake of ease, the rewrite rules will only be represented through transliteration.

Group A includes the general rules of stem adjustment, whereas group B formalizes the processes of assimilation in pattern VIII. In the following lines, we briefly explain the rules of group A, and list the rules of group B.

Group A rules are as follows:

A1 **Loss of the glottal stop in pattern IV**: The first character of i-stem and p-stem of verbs from pattern IV (value 3 in position 3 of the code) is removed.

A2 **Change of *fatha* (*a*) into *damma (u)* in prefix ta- of patterns V, VI and QII**: In passive p-stems, the segment '*a*' of the detransitive morpheme '*-ta*' of patterns V, VI and QII—i.e. the second character of the string—is changed into '*u*'.

A3 ***Fatha* (*a*) insertion in consonant clusters in L-template stems**: Rule for breaking clusters in L-template stems: if a sequence of three consonants—or the vowel lengthening mark E—is found from the end to the beginning, '*a*' is added between the first and second character—counting from the end.

A4 ***Fatha* (*a*) insertion in consonants clusters in H-template stems**: Rule for breaking clusters in H-template stems: if a sequence of three consonants—or the vowel lengthening mark E—is found from the end to the beginning, '*a*' is added between the second and third character—counting from the end.

Rule B1

input	rule	output	gloss
وتَحِد wtaHad	[wy]t -> t~ / ^.?_	تحَّد ttaHad	‘associate’

Rule B2

input	rule	output	gloss
ظنّ Ztanan	t -> ~ / ^[þdðTZ].?_	ظظنّ ZZanan	‘think’

Rule B3

input	rule	output	gloss
زوَّج ztawaj	t -> d / ^.?z_	زدَوَج zdawaj	‘geminate’

Rule B4

input	rule	output	gloss
ضرَّب Dtarab	t -> T / ^.?[DS]_	ضطرب DTarab	‘simmer’

2.2.6 Inflection Module

In the inflection Module, the different stems marking voice and aspect/tense, are passed through their corresponding inflectional paradigm: active and passive p-stems are passed through the perfective paradigm; active m-stem is passed through the imperative paradigm; and active and passive i-stems are passed through imperfective paradigm, and subsequently through the three mood paradigms—indicative, subjunctive and jussive.

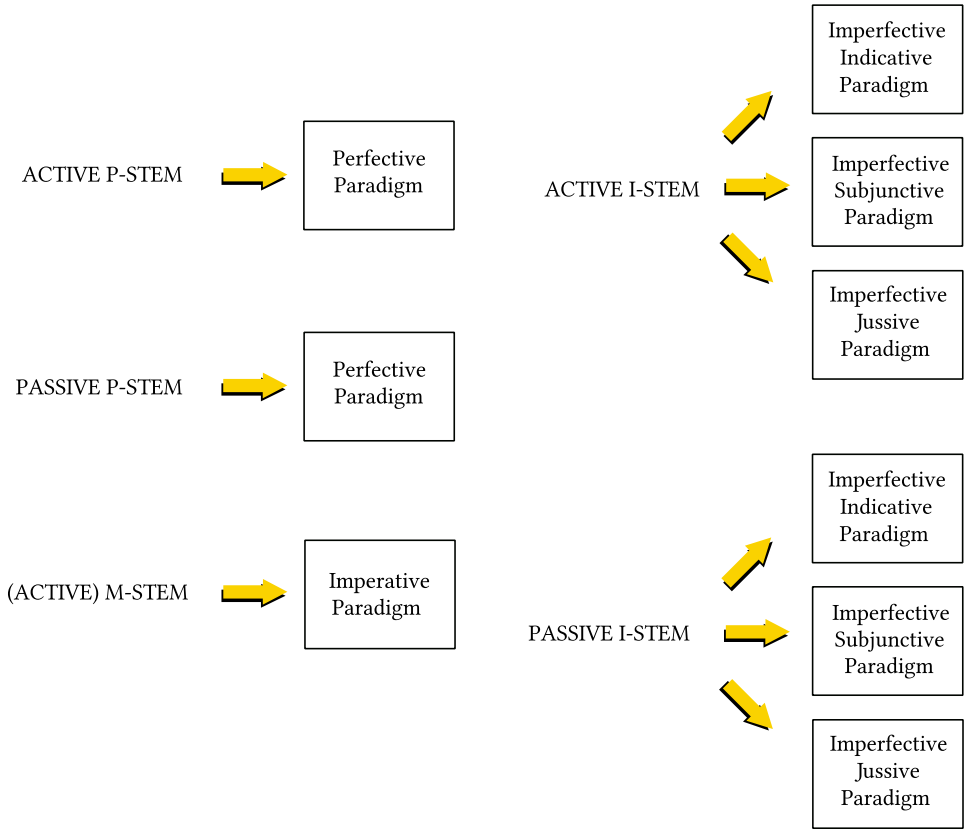


FIGURE 26 Inflection module

All paradigms are shown in the table below. Each column corresponds to one of the paradigms.

TAG	p-stem	i-stem				m-stem
		all	indicative	subjunctive	jussive	
1SN	-تُ	-أ	-ُ	-َ	-	*None*
1PN	Eنَ-	-ن	-ُ	-َ	-	*None*
2SM	-تَ	-ت	-ُ	-َ	-	-
2SF	-تِ	Eِ-ت	-َ	-	-	Eِ-
2DN	Eتُمَا-	Eَ-ت	-ِ	-	-	Eَ-
2PM	-تُم	Eُ-ت	-َ	ا-	ا-	\Eُ-
2PF	-تُنَّ	ت-نَ	-	-	-	-نَ
3SM	-َ	-ي	-ُ	-َ	-	*None*
3SF	-تَ	-ت	-ُ	-َ	-	*None*
3DM	Eَ-	Eَ-ي	-ِ	-	-	*None*
3DF	Eتَا-	Eَ-ت	-ِ	-	-	*None*
3PM	Eُ-	Eُ-ي	-َ	ا-	ا-	*None*
3PF	-نَ	ي-نَ	-	-	-	*None*

TABLE 24 Inflectional paradigms from inflection module

TAG information: 1=first person, 2=second person, 3=third person; S=singular, D=dual, P=plural; M=masculine, F=feminine, N=feminine and masculine (no gender marking). The complete tagset can be found in Appendix C. It is partially inspired by the morphposyntactic tagset proposed by Khoja (Khoja et al. 2003 and Khoja, 2001).

Results

The set of five stems corresponding to each verb will generate the following amount of inflected forms:

STEM	NO. INFLECTED FORMS
active p-stem	13
passive p-stem	13
active i-stem (3 mood paradigms)	39
passive i-stem (3 mood paradigms)	39
active m-stem	5
TOTAL	**109**

Obviously, the system overgenerates. Especially, there are a fairly representative amount of verbs which do not exhibit passive formation. However, within the scope of this project, it is impossible to automatically restrict the passive forms to generate only real ones.

2.2.7 Phonotactic constraints and orthographic normalization Module

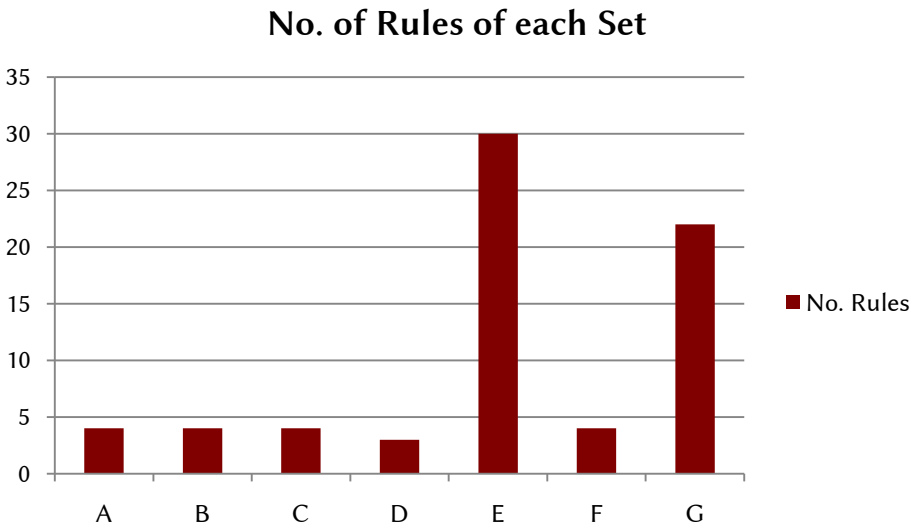
This module receives the regularized inflected forms and applies a sequence of variation rules to obtain the phonological and orthographic—correct—surface form. Until this point all the verbal forms have been generated as if they were *regular*. The aim of this module is to modify the forms in accordance with Arabic phonotactics and orthographic standards to obtain the surface form. In section 2.2.5, we explained the convention used to represent the rewrite rules. The set of rules doing the mapping between the underlying representation to the surface representation are divided into various groups which range from C to G, as we pointed out in section 2.2.5. We have clearly separated linguistic from non-linguistic changes, i.e., phonological from orthographic rules.

Some of the rules must specify conditions to the application of the rules, i.e., it may be that a rule is only applied to imperative forms, so we need to enunciate the rule as "if the form is imperative, then apply the rule". For this specification we need the code associated to each verb lemma and the tag associated to each inflected form—recall that the code and the tagset can be found in Appendix C.

We distinguished six phases in the application of the rewrite rules—group D is divided in two parts. These phases, which must be applied sequentially, are as follows:

Rule Code	Type of rules	No. of rules
C	Orthographic preprocessing: sukun and long vowels	4
D	Initial alif: consonant cluster constraints	2
E	Weak letters alteration rules	30
F	Shadda rules: assimilation of identical consonants	4
D	Initial alif postprocessing	1
G	Hamza normalization rules	22
TOTAL		63

Below, we include a graph comparing the number of rules of each phase. We include the sets of rules A and B, corresponding to the module of stem adjustment. All D rules are placed together.



In the following lines we are going to discuss the main features of each group of rules, indicating if they are phonological or orthographic. Additionally, we will add some of the rules of each phase. We decided not to include all the rules of groups E and G. We considered that listing all the rules without an explanation would be of little value. However, this would be a tremendous amount of work, beyond the scope of this project. Brame (1970) carried out a complete research on the phonological alterations encountered in Arabic morphology, and presented it under his PhD dissertation "Arabic phonology: implications for phonological theory and historic Semitic". This dissertation, which contains research focused exclusively on Arabic phonotactics, gives us an idea of the complexity of the subject, hence our decision to exclude a complete list of rules in this section[79]. The entire list of rules can be found in Appendix E, where the code of the program has been included. In the program, rules are formalized as regular expressions. On the right of each regular expression

[79] Unfortunately, we discovered this work after having formalized the rules of phonotactics. It would be very advantageous to examine the rules described by Brame and compare them with our own rules. Yet, this is a work that requires much additional effort, and it is beyond the scope of this dissertation.

there is a comment containing the rule in the form of a rewrite rule. The rewrite rule is represented according to the convention we described in the introduction to section 2.2.5.

2.2.7.1 Orthographic preprocessing: sukun and long vowels

At this stage the sukun is added to all unvocalized consonants, so that the resulting form is fully diacritized. We refer to this rule as *sukunizer*. Additionally, the symbol 'E', which is used to represent vowel lengthening in our program, is substituted by a long vowel character, either *waw*, *alif* of *ya*, depending on the preceding short vowel.

C1 *sukunizer*: adds sukun in all contexts where it is expected

C2 aE -> aA

C3 uE -> uw

C4 iE -> iy

2.2.7.2 Initial alif: consonant cluster constraints

A segment '*Au*' or '*Ai*' is added at the beginning of initial consonant clusters to break this prohibited segment. C represents any consonant. By the conversion of an empty set into an specific segment, we attempt to represent a rule of insertion instead of a rule of substitution, but without changing the notation.

D1 ∅ -> Au / ^_C·C~?u

D2 ∅ -> Ai / ^_C·C~?[^u]

2.2.7.3 Weak letters alteration rules

The so-called weak letters *waw* and *ya* are responsible for the majority of the phonological alterations. The rules which handle these segments mainly include cases of elision and substitution, which reflect assimilation processes. In total, this set of rules is composed of 30 rules. From these, 21 have conditions which restrict the application of the rules. We think that the number of conditioned rules is high because there are always a few cases where a form containing weak letters performs regularly. In general, this happens with underused forms. However, the restrictions presumably affect a small set of forms from the complete lexicon.

We are going to discuss one example of non-conditioned rule and one example of conditioned rule. In each example, we add the python code used to codify that rule—recall that there is a visualization problem in the regular expressions making the understanding of the string almost impossible. Then, the rule is explained through the example.

form=re.sub('.ُوِ)',r'1\ِي',form) # E12 uwi -> iy / _Ca

Rule E12	**unconditioned rule**		
input	**rule**	**output**	**gloss**
قُوِلَ quwila	uwi -> iy / _Ca	قِيلَ qiyla	“it is said”

Rule E12 deals with the sound *wi*. This segment is discouraged by the Arabic phonological system. Hence, the rule handles the transformation of this sound into a more harmonic sound *iy*. The context specified by the rule indicates that the *pattern* must be followed by a consonant plus a vowel *a*, so that the rule is applied. In the example, we can see the perfective passive formation of the common verb قال *qaAla √qwl*. The regularly generated passive **quwila* is substituted for the more melodious sound *qiyla*.

```
if ((d_code['Vocalization']['Perf V2']!='1')|(d_code['Vocalization']['Imperf V2']!='1')):
    form=re.sub('([يو]([وي][[ٔ?]يّ-ء)',r'\1\2',form)          # E19  [ui][wy] -> Ø / C~?_iy
```

E12	**conditioned rule: applies to all forms except Iuu verbs**		
input	**rule**	**output**	**gloss**
يَدْعُوِينَ yad·çuwiyna	[ui][wy] -> ∅ / C~?_iy	يَدْعِينَ yad·çiyna	“she invites”

In the first line of the code, the condition is specified; in the second line, the rule itself is formulated. The rule is in fact the same as in the previous case: the segment wi is discouraged. In this case the result of the rule is the loss of the segment. In the example, we use the verb داع *daAça √dwç*. The condition is simply stating that the rule must be applied to all forms except those of pattern Iuu verbs. Generally, pattern Iuu verbs are few and rarely used. It would be very interesting to study in more detail why these verbs reject the phonotactic prohibition of the segment wi. It also happens in other conditioned rules that the rule is applied to all forms but a small set of rare forms. In general, forms that have been rarely used, tend to exhibit completely regular behaviours. Presumably, as the rate of use is low, they tend to be more regular. We have enunciated the condition as “if the form does not belong to pattern Iuu verbs, then apply the rule”. The condition has to be transmitted through a restriction in the code we associated to each verb lemma. Iuu verbs correspond to code 00L0101—in Appendix D we have the pattern-code equivalences. The code is distinguished from the rest by the two number 1s. The first 1 indicates that the verb exhibits a vowel *u* as the thematic vowel of perfective and , the second one, indicates that it takes *u* as the thematic vowel of the imperfective—both correspond to the second vowel of the form. Hence, the condition is as follows: “if the second vowel of the perfective is not 1 and also if the second vowel of the imperfect is not 1, then apply the rule. We employed De Morgan’s law to make the condition clearer—the negation of a conjunction is the disjunction of the negations, so ¬(A∧B) = ¬A∨¬B. Finally, the condition is expressed

as “if the second vowel of the perfective is not 1 or if the second vowel of the imperfect is not 1, then apply the rule”.

2.2.7.4 Shadda rules: assimilation of identical consonants

These set of rules deal with cases of assimilation in which the vowel between two identical consonants is lost or moved. In any case, the identical vowels are placed together. Orthographically, the main characteristic of these rules is that a shadda must be included in the string to represent the presence of the two-consonant sequence. One of the 4 rules is orthographic. In this case, the two identical consonants are next to each other—a sukun separates them. We are going to show an example of this orthographic rule. In this case, we do not include the python code, as we do not feel it necessary to include it in every example—recall that the complete python code of all the program for verbal generation can be found in Appendix E.

Rule F4

input	**rule**	**output**	**gloss**
أَثْبَتْتُ Áaþ·bat·tu	·C^1 -> ~ / C^1_v^1 (C stands for consonant)	أَثْبَتُّ Áaþ·bat~u	“I proved”

The rule simply states that the two letters are fusioned into one, and a shadda is included to indicate the presence of a double consonant.

2.2.7.5 Initial alif postprocessing

This isolated rule is connected with rules D1 and D2. These rules inserted an alif plus a vowel to the beginning of forms starting with consonant clusters. However, some of these forms were modified by the set of weak rules, and the consonant clusters were broken up. So at this stage we need a rule to remove the affix plus vowel segments which was erroneously added.

D3 Av -> ∅ / ^_Cv

2.2.7.6 Hamza normalization rules

This set of rules are purely orthographic. All the rules deal with the problem of the hamza letters. There is a total of 20 rules, none of which is conditioned. Below, we show three examples of the application of rules. As in all phases, the input forms are outputs of the previous modules.

Rule G4

input	rule	output	gloss
أَءْخُذُ Áac·xuða	[cÁÀúý]a[cÁÀúý]· -> Ã	آخُذُ Ãxuða	“I take”

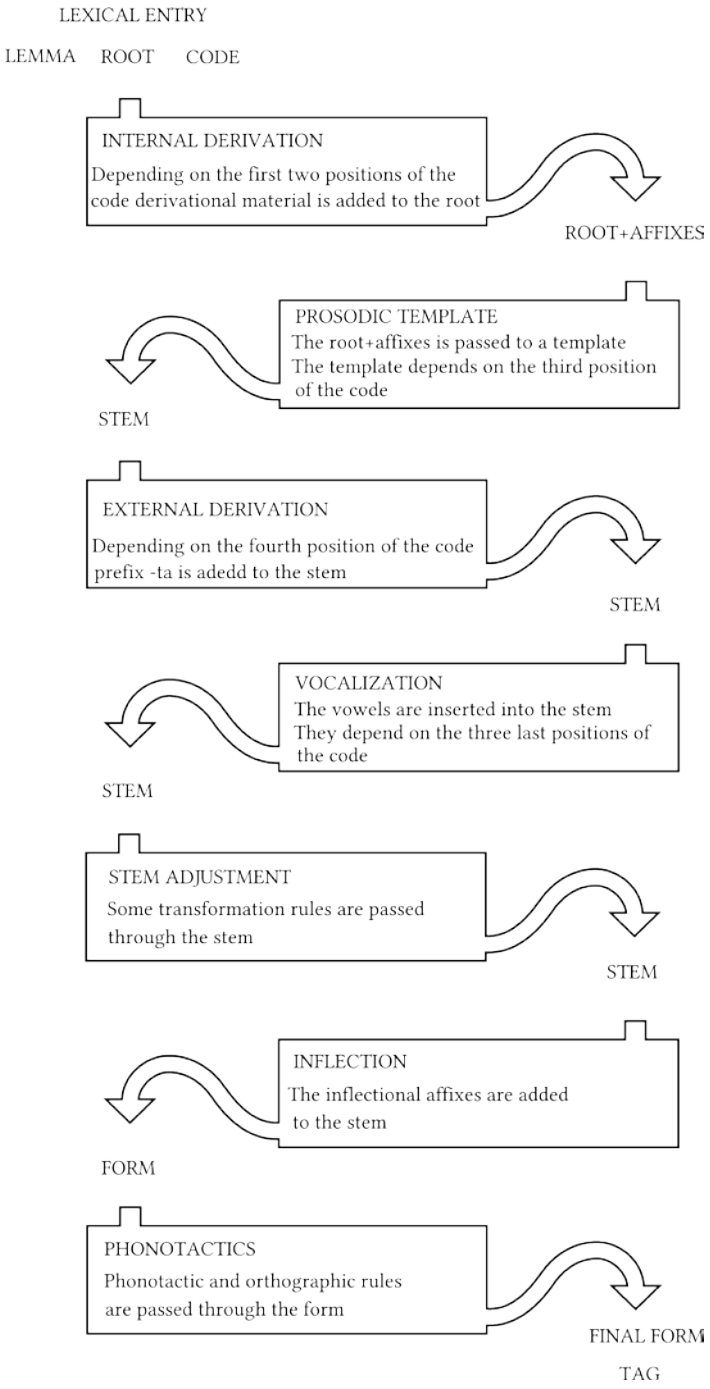
Rule G15

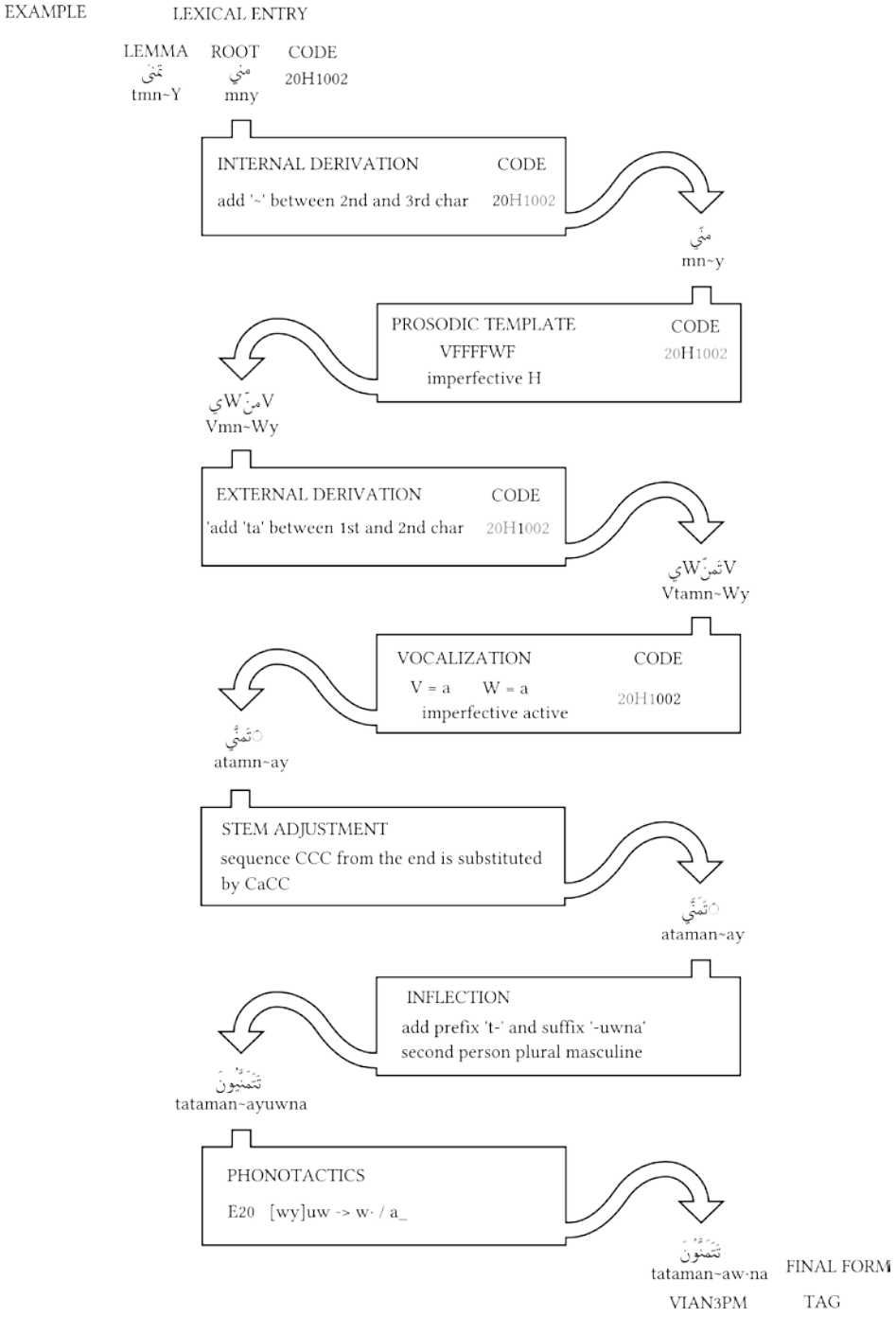
input	rule	output	gloss
يَقْرَءُونَ yaq·racuwna	[cÁÀý] -> ú / [aA]_u.	يَقْرَؤُونَ yaq·raúuwna	“they (pl.) read”

Rule G7

input	rule	output	gloss
تَخْطِءِينَ tux·Ticiyna	[cÁÀú] -> ý / i_	تَخْطِيِنَ tux·Tiýiyna	“you (fem.) make”

2.3 General framework of the computational system




EXAMPLE
LEXICAL ENTRY
LEMMA
تَمَنَّى
tmn~Y
ROOT
مني
mny
CODE
20H1002
INTERNAL DERIVATION
add '~' between 2nd and 3rd char
CODE
20H1002
مَنَّي
mn~y
PROSODIC TEMPLATE
VFFFWF
imperfective H
CODE
20H1002
يW مَنَّ V
Vmn~Wy
EXTERNAL DERIVATION
'add 'ta' between 1st and 2nd char
CODE
20H1002
يW تَمَنَّ V
Vtamn~Wy
VOCALIZATION
V = a W = a
imperfective active
CODE
20H1002
تَمَنَّي
atamn~ay
STEM ADJUSTMENT
sequence CCC from the end is substituted by CaCC
تَمَنَّي
ataman~ay
INFLECTION
add prefix 't-' and suffix '-uwna'
second person plural masculine
تَتَمَنَّيُونَ
tataman~ayuwna
PHONOTACTICS
E20 [wy]uw -> w· / a_
تَتَمَنَّوْنَ
tataman~aw·na
VIAN3PM
FINAL FORM
TAG

3 Evaluation

Logically, the degree of accuracy achieved by a system has a contribution to real life applications (El-Beltagy and Rafea, 2009), thus it is basic for the system's usefulness to have a highly reduced amount of failures. For this reason, the evaluation is an essential part of every morphological system.

Precision and recall are two indexes commonly used to measure the accuracy of a lexical database. Precision indicates the relevant tokens retrieved from a dataset and recall indicates if there are no tokens missed (Andersen, 2010). In our case, precision shows the amount of correct forms which have been generated and recall shows if all the forms available in the Gold Standard—the lexical database used to evaluate our results—are included in the Jabalín lexicon as well.

In sum, the evaluation will allow us to analyze the kind of errors produced by the system and its empirical coverage.

3.1 The lexicon of inflected verbs ElixirFM

We carried out an evaluation of the generated lexicon of inflected forms against a list of verbs extracted from the morphological analyzer ElixirFM (Smrž, 2007). We assumed that the lexicon extracted from the ElixirFM software is a validated database of Arabic conjugation, so we consider it a gold standard. We based this assumption on the fact that ElixirFm is an improvement on the BAMA analyzer, which has reportedly achieved 99.25% precision (Rodrigues and Cavar, 2007). However, no recall was reported. Starting from this assumption we normalized the ElixirFm lexicon, so

that it shares a common format with our lexicon. After that, we compared the reference lexicon with our generated lexicon and searched for each of our verbal entries in the reference lexicon, obtaining a number of successes and failures.

We used the different functionalities of the ElixirFM software to extract a lexicon of inflected verbs. Each entry of the lexicon had the following information: the grammatical tag according to the ElixirFM convention, the root, internal information about the inflectional properties of the form, the inflected form, the pattern of the verb—in the western tradition form—and a coordinate which serves as an identifier for each verb.

Tag	Form	Root	Inflectional_info	Pattern	Coordinate
VIIA-3MS--	يُفَكِّر	"ف ك ر"	"yu" >>| FaCCiL |<< "u"	II	(4831,1)

In our generated lexicon, as we have shown previously, the information for each entry has this information:

Tag	Form	Root	Lemma	Code
VIAN3SM	يُفَكِّر	فكر	فَكَّر	20H0010

To make the comparison, we keep the form and root values from both lexicons. In a second step, we establish an equivalence between the tagset of the reference lexicon and our tagset. ElixirFM includes 6 grammatical categories: tense/aspect (perfective, imperfective, imperative), voice (active, passive), mood (indicative, subjunctive, jussive, energetic[80]), person (first, second and third), gender (masculine and feminine) and number (singular, dual, plural). Each verb has 192 theoretically possible inflected forms. Our classification does not include the energetic mood, as it is scarcely found

[80] Called by the grammarians النون المؤكّدة the *corroborative nun* (Wright, 2007).

in Arabic texts, even old ones. Besides, when the form does not show distinction in gender, as in first person or in second person dual, it is marked as neuter—in the sense that it matches both masculine and feminine genders. The first person is not inflected for form as well, so number may only be singular or plural. The absence of energetic mood and the feature syncretism of some forms in our category system, gives us a smaller set of possible inflected forms than the ElixirFM, a total of 109 theoretical forms for each verb. Thus, in order to carry out the evaluation, the ElixirFm tagset has been reduced and normalized with our tagset[81], so that each verb has 109 forms—the rest of the forms in the lexicon will be ignored. The Jabalín tagset and the ElixirFm tagset are compared in Appendix C.

The lemma and the code information are deduced from the reference lexicon—the lemma is taken from the corresponding perfective third person singular form of the verb. The code, in turn, is established using the pattern information and the additional inflectional information, in case the pattern is not explicit enough [82].

[81] The correspondence of the two tagsets is found in Appendix 4.

[82] Pattern I corresponds to six different codes in our model. Moreover, patterns I to IV of the quadriliteral verb are not differentiated from the same patterns of the triliteral verbs. All this information must be disambiguated.

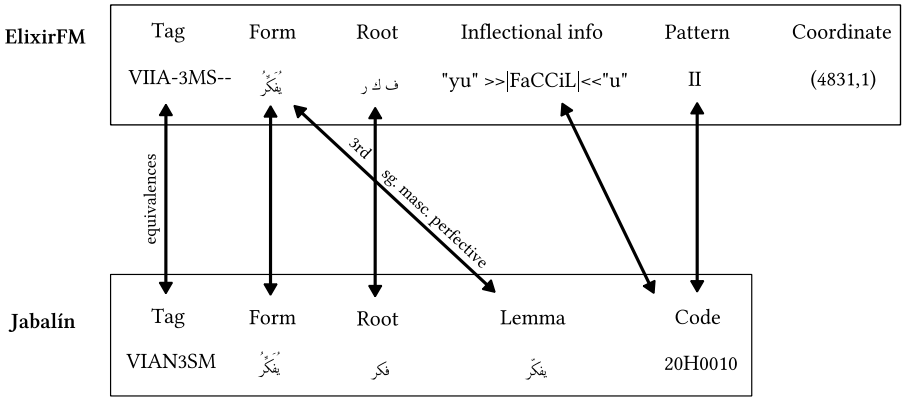


FIGURE 27 Translation of information from ElixirFm to Jabalín

	No. Lemmas	No. Forms	Forms per lemma
ElixirFM[83]	9009	1,752,848	192
Jabalín	15,452	1,684,268	109
Common	6878	749,702	109

TABLE 25 Data on number of lemmas and forms in ElixirFM and Jabalín

When the reference lexicon was normalized and the equivalences established, the evaluation was carried out—first, all the information associated to the inflected form is searched in the reference lexicon and, if it is found, then the evaluation is possible, so both forms are compared; if it is not found, the form is not evaluable. We found that 44% of our lexicon is evaluable.

[83] Sometimes, in the ElixirFM lexicon there is more than one form corresponding to the same tag, so the total number of forms does not equal the number of lemmas plus the number of forms per lemma.

	No. forms	%
Total	1,684,268	-
Eval	749,051[84]	44%
Not eval	935,217	56%

TABLE 26 Evaluated and non-evaluated data

In relation with the comparison of the lemmas of both lexicons, the ElixirFM has 9,009 lemmas, whereas our lexicon has 15,452. Both lexicons have 6,878 lemmas in common. However, there are 2,131 lemmas only present in ElixirFM and 8,581 only present in Jabalín. This means that we have a low recall rate with respect to the ElixirFM database. Even though both gaps may seem substantial, we believe that it is an inherent problem of working with Classical Arabic lexicon and, ultimately both ElixirFM and Jabalín include a high percetage of obsolete lexical entries.

From the 749,051 forms which are evaluable, 99.52% of them are correct, whereas 0.48% are incorrect, i.e. the conjugational information of the form is found in the reference lexicon, but the form does not match.

	No. forms	% from total	% from eval
Correct	745,436	44,26%	99.52%
Incorrect	3,615	0.21%	0.48%
No data	935,217	55.53%	-
Total	1,684,268	-	-

TABLE 27 Results from the evaluation (precision)

[84] 651 forms from the total number of evaluable forms, i.e. 749,702, were not evaluated because some discrepancies were found in grammar books. This means that the total number of evaluated forms was reduced to 749,051.

3.2 Buckwalter and Parkinson's most frequent verbs

In addition to the evaluation against the ElixirFM verbal lexicon, we have compared our generated lexicon with a list of the 825 most frequent verbs taken from "A frequency dictionary of Arabic. Core vocabulary for learners" (Buckwalter and Parkinson, 2011). The list combines two different lists included in the dictionary: a list of the most frequent verbs of each conjugational type and a separate list of the verbs in the 5,000 most frequent words. The top 5,000 most frequently used words, which composes the content of the dictionary, were extracted from a 30-million-word corpus of MSA and Arabic dialects (10% spontaneous speech and 90% written). The dialectal varieties include Egyptian, Levantine, Iraqi, Gulf, and Algerian. The written data contains 5 parts of about 5.4 million words each: (1) daily newswire, (2) newspaper texts, (3) academic works, (4) postings on Internet discussion forums, and (5) literature texts.

First, the corpus data was processed using the Buckwalter morphological analyzer, so each lexical item was given all its possible analyses. The disambiguation and tagging tasks were carried out with an online interactive concordance and annotation tool developed by Brigham Young University. The most frequent 5,000 lemmas were computed adjusting the frequency data to their dispersion rate.

The list of 825 frequent lemmas was compared with our list of lemmas. A small amount of lemmas were absent in ours, so they were included afterwards. All the lemmas were checked in our generation, achieving a precision of 99.27% correctness. Below, we include a list with all the verbs from the list which were not included in our lexicon and have been added.

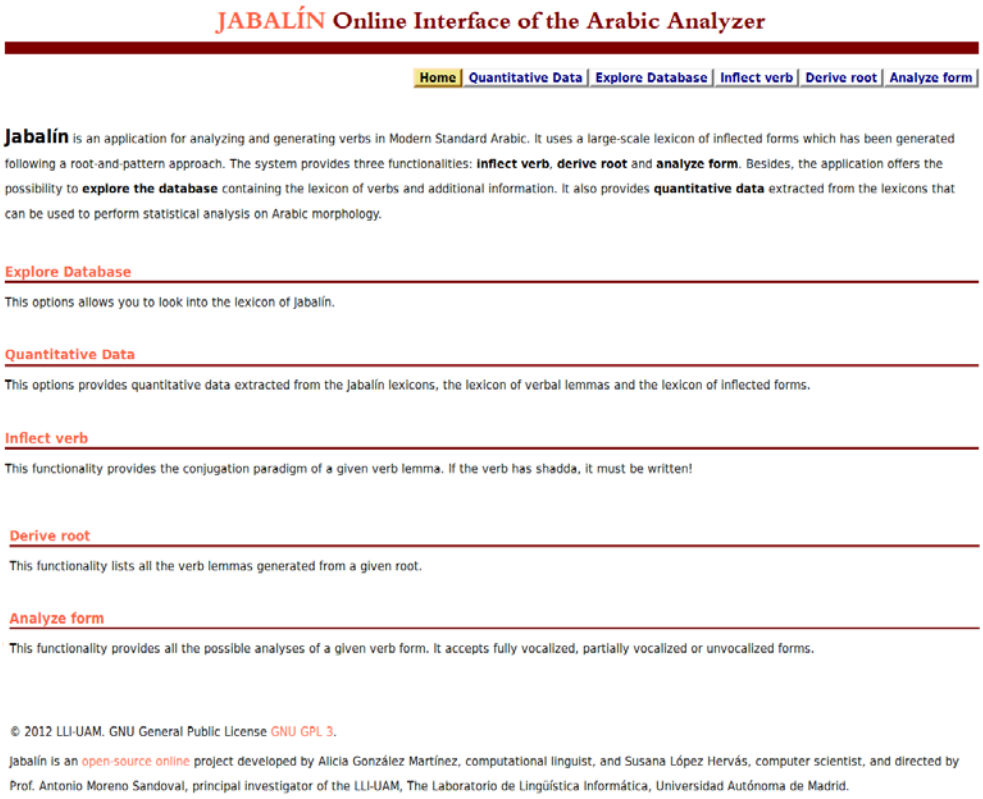
lemma	root	code
صلّح	صلح	20H0010
نمّ	نمم	00L0001
نمّ	نمم	00L0000
نوّه	نوه	20H0010
تركّز	ركز	20H1002
تجزّأ	جزء	20H1002
حدّد	حدد	20H0010
خدم	خدم	00L0000
خفي	خفي	00L0202
قفر	قفر	00L0000
سرى	سري	00L0000
حيّ	حيي	00L0002

lemma	root	code
لاس	لوس	00L0001
ليس	ليس	00L0000
آه	ءوه	00L0001
أبى	ءبي	00L0002
لبّى	لبي	20H0010
انراد	رود	01L0000
تلفن	تلفن	00H0010
دردش	دردش	00H0010
مسخر	مسخر	00H0010
تعلمن	علمن	00H1002
تمسخر	مسخر	00H1002

TABLE 28 Verbs from Buckwalter and Parkinson which were absent in Jabalín

4 Jabalín Online Interface

The Jabalín Online Interface is a web application for analyzing and generating Arabic verbs. It uses the lexicon of inflected verbs provided by the generation system described in previous sections. The online interface is hosted at the LLI-UAM laboratory web page, under the address http://elvira.lllf.uam.es/jabalin/.



Results

The interface provides four functionalities: explore database, inflect verb, derive root and analyze form.

Explore database allows one to look into the lexicon of Jabalín. It includes information about all the inflected forms from the lexicon: the fully vocalized form, the form without diacritics, the lemma, the root, the code of the verb, the tag containing the inflectional information and data from the evaluation against the ElixirFM lexicon—if the specific form was evaluated, the field *eval* shows the value *yes*.

JABALÍN Online Interface of the Arabic Analyzer

Home | Explore Database | Inflect verb | Derive root | Analyze form

id: 20000 of **1684377** total forms Update

◄ back | next ►

id	form	vocalized_form	lemma	root	code	tag	translated_tag	eval
20000	تستأسر	تَسْتَأْسِرْ	استأسر	ءسر	04H0000	VIAY3SF	imperfective active jussive third person singular feminine	YES
20001	تستأسر	تَسْتَأْسِرْ	استأسر	ءسر	04H0000	VIAY2SM	imperfective active jussive second person singular masculine	YES
20002	تستأسروا	تَسْتَأْسِرُوا	استأسر	ءسر	04H0000	VIAY2PM	imperfective active jussive second person plural masculine	YES
20003	تستأسرا	تَسْتَأْسِرَا	استأسر	ءسر	04H0000	VIAY2DN	imperfective active jussive second person dual	YES
20004	تستأسرا	تَسْتَأْسِرَا	استأسر	ءسر	04H0000	VIAY3DF	imperfective active jussive third person dual feminine	YES
20005	تستأسري	تَسْتَأْسِرِي	استأسر	ءسر	04H0000	VIAY2SF	imperfective active jussive second person singular feminine	YES
20006	يستأسروا	يَسْتَأْسِرُوا	استأسر	ءسر	04H0000	VIAY3PM	imperfective active jussive third person plural masculine	YES
20007	يستأسرا	يَسْتَأْسِرَا	استأسر	ءسر	04H0000	VIAY3DM	imperfective active jussive third person dual masculine	YES
20008	تستأسرن	تَسْتَأْسِرْنَ	استأسر	ءسر	04H0000	VIAY2PF	imperfective active jussive second person plural feminine	YES
20009	نستأسر	نَسْتَأْسِرْ	استأسر	ءسر	04H0000	VIAY1PN	imperfective active jussive first person plural	YES
20010	يستأسرن	يَسْتَأْسِرْنَ	استأسر	ءسر	04H0000	VIAY3PF	imperfective active jussive third person plural feminine	YES
20011	يستأسر	يَسْتَأْسِرْ	استأسر	ءسر	04H0000	VIAY3SM	imperfective active jussive third person singular masculine	YES
20012	أستأسر	أَسْتَأْسِرْ	استأسر	ءسر	04H0000	VIAY1SN	imperfective active jussive first person singular	YES
20013	يستأسرون	يُسْتَأْسَرُونَ	استأسر	ءسر	04H0000	VIPN3PM	imperfective passive indicative third person plural masculine	YES
20014	تستأسران	تُسْتَأْسَرَانِ	استأسر	ءسر	04H0000	VIPN2DN	imperfective passive indicative second person dual	YES
20015	تستأسر	تُسْتَأْسَرُ	استأسر	ءسر	04H0000	VIPN3SF	imperfective passive indicative third person singular feminine	YES
20016	أستأسر	أُسْتَأْسَرُ	استأسر	ءسر	04H0000	VIPN1SN	imperfective passive indicative first person singular	YES
20017	تستأسرون	تُسْتَأْسَرُونَ	استأسر	ءسر	04H0000	VIPN2PM	imperfective passive indicative second person plural masculine	YES
20018	نستأسر	نُسْتَأْسَرُ	استأسر	ءسر	04H0000	VIPN1PN	imperfective passive indicative first person plural	YES
20019	يستأسران	يُسْتَأْسَرَانِ	استأسر	ءسر	04H0000	VIPN3DM	imperfective passive indicative third person dual masculine	YES
20020	تستأسرين	تُسْتَأْسَرِينَ	استأسر	ءسر	04H0000	VIPN2SF	imperfective passive indicative second person singular feminine	YES
20021	يستأسر	يُسْتَأْسَرُ	استأسر	ءسر	04H0000	VIPN3SM	imperfective passive indicative third person singular masculine	YES
20022	تستأسران	تُسْتَأْسَرَانِ	استأسر	ءسر	04H0000	VIPN3DF	imperfective passive indicative third person dual feminine	YES
20023	تستأسرن	تُسْتَأْسَرْنَ	استأسر	ءسر	04H0000	VIPN2PF	imperfective passive indicative second person plural feminine	YES
20024	تستأسر	تُسْتَأْسَرُ	استأسر	ءسر	04H0000	VIPN2SM	imperfective passive indicative second person singular masculine	YES

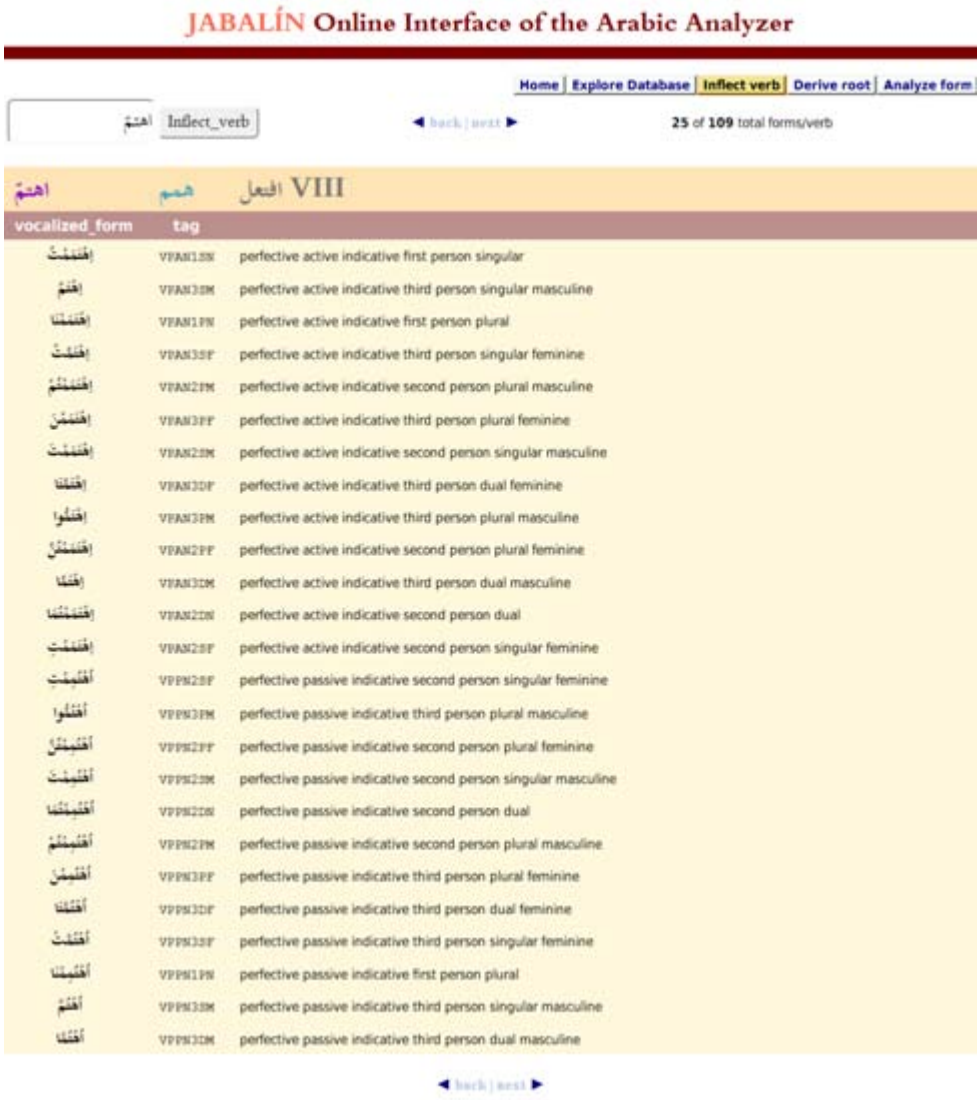
109 forms/lemma
15453 lemmas
3708 roots

741963 evaluable forms
735232 correct forms
99.09 %correct forms

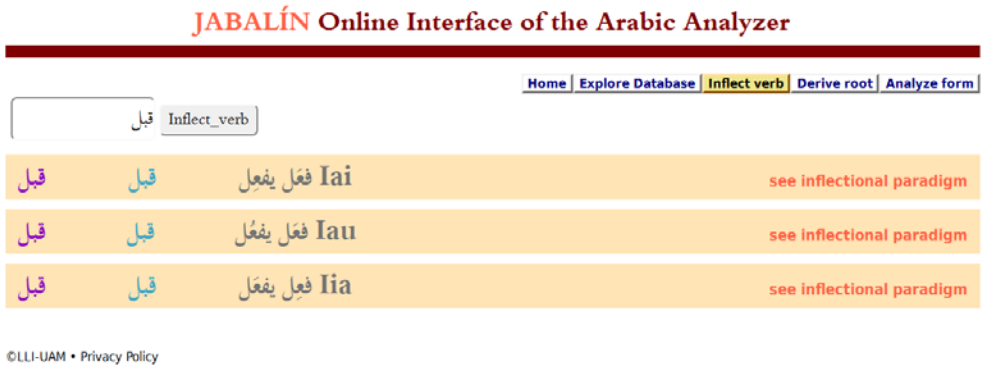
◄ back | next ►

On the bottom left of the page some basic quantitative data on the lexicon and on the evaluation given. On the left part the number of forms per lemma is written—i.e. the size of the conjugational paradigm—the total number of lemmas and the total number of roots. On the right, the total number of forms which have been evaluated against the ElixirFM lexicon is included, the number of verbs found to be correct and the total percentage of correct forms.

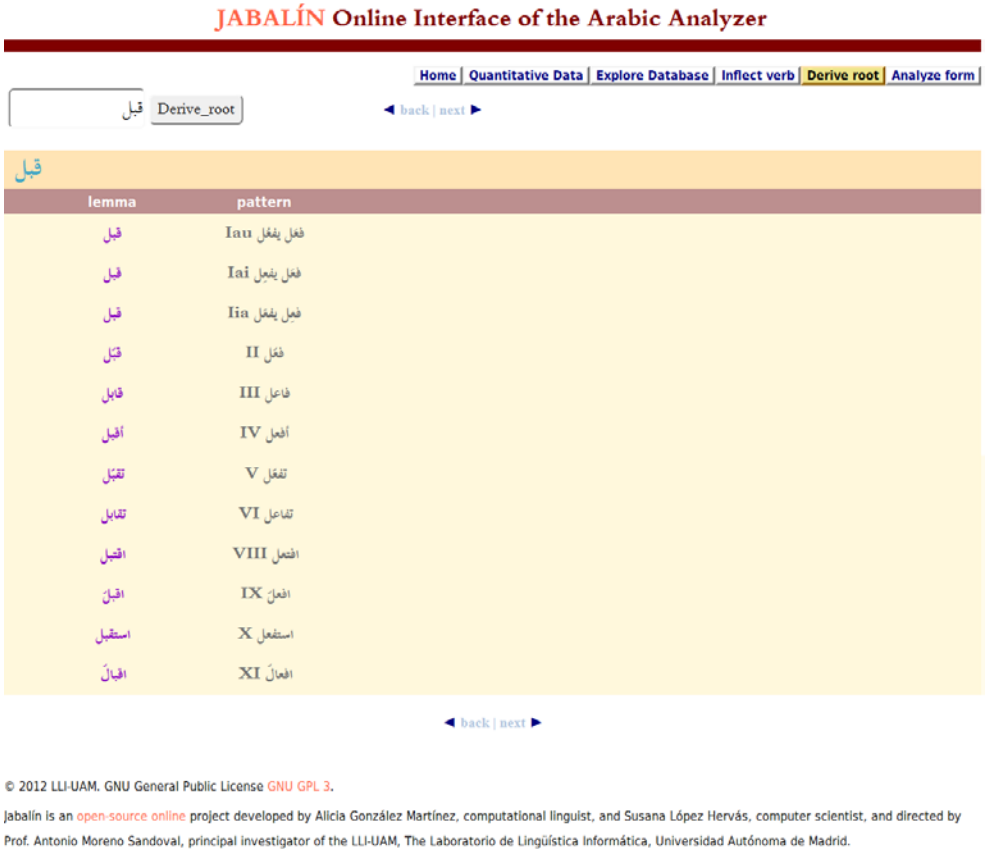
Inflect verb provides the conjugation paradigm of a given verb lemma. It includes the root and the pattern of the verb—in Arabic and in Roman numerals convention—the fully focalized form and the inflectional tag.



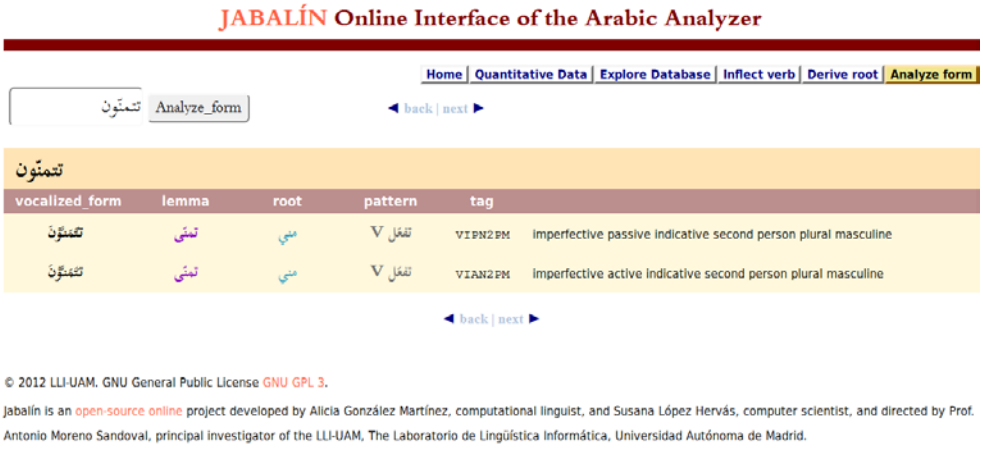
When the lemma entered corresponds to more than one verb, the corresponding verbs are listed and the user must select the search option.



Derive root lists all the verb lemmas generated from a given root and its corresponding patterns.



Analyze form provides all the possible analyses of a given verbal form. It accepts fully vocalized, partially vocalized or unvocalized forms.



Discussion

We have defined two main objectives for this project—to formalize an elegant linguistic description of Arabic verbs and to develop a set of computational resources making use of this linguistic formalization. We have created a lexicon of verbal lemmas and, subsequently, we developed a verbal generation system which provides a lexicon of inflected forms. The lexicon of inflected forms has been validated by means of an extensive evaluation against the ElixirFm lexicon. Likewise, it has been used to create a morphological analyzer and generator for Arabic verbs which includes an online interface. All of these tasks are part of the Jabalín project of Arabic computational morphology. The project also includes a transliteration system and a module for extracting quantitative data from the lexicons, which will be used to perform linguistic analysis in the future—this additional work is included in the appendices.

The implementation of the verbal generation system is in turn used to validate the accuracy of the linguistic description. In this respect, Farghaly states that "symbolic NLP has often been used to test the validity and accuracy of grammars that linguistics develop" (Farghaly, 2012)—Kiraz (1999) holds a similar opinion. Yet this idea must be taken with caution, as we cannot assert that a *working* formalization shows the actual structure of a language in a cognitive sense. It simply means that the analytical model on which the program is based enables a formal description of that language—of course, if the implementation has been evaluated and achieves a high precision—i.e. it describes a system of organized and interrelated units and operations which emulate successfully the grammar of the language. However, we believe that if the formalization is simple and precise, and its operations are highly universal—in that language—the formal model has some significance in descriptive linguistics, at least it can offer a hint as to some of the linguistic features operating in the grammar. For instance, our classification of verbs does not have to make any distinction between

verbs built upon triliteral or quadriliteral roots—contrary to traditional grammar description which is followed by many computational models—therefore we propose to consider both types as members of the same conjugational class.

With regards to the Arabic writing system and its computational treatment, we have benefit from all the orthographic resources provided by the Arabic script in its full form. Arabs have supplied their writing system with a consistent set of disambiguation tools shortly after the advent of Islam. Indeed, Arabic diacritics were invented with the purpose of functioning as a device to disambiguate the script, which could be quite misleading. Hence, by generating a fully-diacritized lexicon, we keep relevant information which will be essential in a future disambiguation stage.

Within the program of verbal generation, we decided to work directly with Arabic characters, instead of using a transliteration. We based our choice on the fact that nowadays Unicode codification is easily handled by many programming languages and platforms. Python version 3 is one of the various choices, for it does not require any special, explicit feature to handle Arabic characters. However, we encountered a problem which, to our knowledge, has not been solved yet—the problem of mixing directionalities. When characters meant for different directions are mixed together, it is impossible to find a proper visualization for the resulting string. This problem was felt squarely in writing the regular expressions representing the rewrite rules in the module of phonotactic constraints and orthographic normalization. The special characters used within the expressions follow a left-to-right direction, while Arabic characters follow a right-to-left direction. As we had already started the implementation, we decided to keep on working with Arabic script. After some training, it is possible to work efficiently despite the directionality problem. However, the complexity derived from working with two-direction characters could well have been replaced by using a transliteration.

On the other hand, working with a transliteration may not be desirable, due to the inherent problem of finding a suitable transliteration. In this sense, Al-Sughaiyer and Al-Kharashi stated that "writing rules and lexicons in Roman rather than Arabic characters may not be advisable, especially in the long term. The reason is that there is no universally accepted standard for representing Arabic characters and symbols in English" (Al-Sughaiyer and Al-Kharashi, 2004). It is true that in recent years Buckwalter's transliteration has turn into a fairly used default standard. However, as we explain in Appendix A, we found various drawbacks with this transliteration—although we believe it to be the best among all proposals—and for this reason we decided to develop a new one. Our transliteration, although not used in the program of verbal generation itself, is used in the comments of the regular expressions to show a clear representation of the rules, as well as in the written parts of this dissertation.

Our model of verbal generation is based on a multi-level representation of the stem and a classification of prosodic structures. Such approach benefits from both the most recent analyses in the field of Semitic linguistics—especially McCarthy (1981; 1990; 1993; 1996), Watson (2007), Danks (2011) and Cahill (1990, 2007, 2010)—and of the traditional interpretations and observations of the classical Arab grammarians, especially in the studies of al-Khalil on quantitative prosody. We believe that our formalization presents a manageable and straightforward description of Arabic verbal morphology.

McCarthy was the first to propose an analysis of Arabic morphology based on prosodic principles from a contemporary perspective. In his first works, he was highly influenced by the theory of autosegmental phonology developed by Goldmith. The most relevant idea of the proposal presented by McCarthy was to analyze morphological structures at different levels—a root, a vocalism and a syllabic skeleton for each of the derived patterns. Watson and Danks carried out deep morpho-phonological studies from this perspective and various computational models use this

analysis—for instance, the model proposed by Kiraz, or the analyzer Magead, developed by Habash.

The analysis we propose follows this line of research too. Verbal stems are composed of the intertwined combination of a consonantal root, a vocalism and optional affixational processes. These three elements are superimposed on a prosodic template by means of a straightforward ordering algorithm. Perhaps the major achievement of our classification is that there are only two classes of prosodic templates for all verbs—including all rare patterns present in Classical Arabic.

Our model sets the root-morpheme in a prominent position—each verbal form is generated starting from the root consonants. In a second step additional derivational material—what we call the affixational processes—is inserted. The resulting amalgam is placed on a prosodic template and covered with a specific vocalic melody to create the different stems. Each verb entry has three stems: p-stem (perfective), i-stem (imperfective) and m-stem (imperative).

Cahill has also proposed an analysis of Arabic morphology based on syllabic structure. Her theory is influenced by previous analyses developed around the ablaut process typical of European languages. Strikingly, we arrived to similar conclusions about the core importance syllabic structure has in Arabic grammar.

Likewise, we fully endorse Cahill's view about surface morphological abnormalities: "(...) while Arabic weak roots are often cited as behaving differently morphologically, we argue that they behave entirely regularly morphologically, but their behaviour is determined by their phonology" (Cahill, 2010). As we will explain below, we believe that this idea has been demonstrated by means of implementing the module of phonotactic constraints in the generation system, with a high degree of probability.

Results

There is an remarkable system of subregularities operating in the formation of phonological alternations. We described 4 types of alterations, 3 of them phonological and 1 orthographic:

1. the form has a weak root, i.e., it contains the semiconsonants w or y
2. the form has a geminated root , i.e., the second and the third radicals of a triliteral root are the same
3. the form has the derivational affix –t-, which corresponds to pattern VIII
4. the form has a hamzated root, i.e., it contains the glottal stop c—there are six letters to represent it

The module of phonotactic constraints and orthographic normalization—in Appendix E we find the entire set of rules within code—deals with the alterations listed above, which handle the distance between the underlying representation and the surface representation. Each alteration is formalized in the form of a rewrite rule—they are treated in detail in Results-2.2.5. The rewrite rules as follows:

a -> b / _c

The majority of the rewrite rules are applied to all the underlying forms generated by the system, i.e. all forms must pass through these rules. However, some rules have conditions which restrict the application of the rules to a subset of forms.

General rule:

x -> y / _a

Conditioned rule:

If the form is imperative, then apply x => y / _a

General rules and conditioned rules have different linguistic implications for the overall model: in the case of general rules, as all forms must pass through these rules, they cannot be morphologically motivated, but instead they are caused by phonological constraints. On the other hand, conditioned rules are subject to three possible interpretations:

a. they may be connected to morphological constraints,
b. they may be lexical exceptions,
c. they may be the result of a misleading formalization; the context delimited by the alteration is not restricted enough, and it is in turn specified by indicating a condition to the wordform for the rule to be applied

Case (a) is the most probable explanation if the amount of wordforms affected by the rule is high. On the contrary, if the wordforms affected by the rule are few, the probable reason is that case (b) occurs. In turn, we assume that case (c) is very rare, as it can only be possible if the context we must specify is only present in a set of wordforms corresponding to a morphological class, but without morphological conditioning.

It is important to note that exceptions are assumed to be normal and expected in the description of any language. This, added to the fact that the majority of the word-forms are coupled just with general rules and the conditions affect a reduced set of forms, is presumably a reason to believe that conditioned rules are generally dealing with lexical irregularities.

Coming back to the underlying forms, we discovered an astonishing regularity in the formation of the verbal stem. We saw that prosodic structures are grounded in the principle of a moraic opposition of the light type versus heavy syllables. The structures were reduced to two possibilities: in one group of verbs the second syllable starting from the end is light, in the other, this syllable is heavy. As we have already

pointed out, the relevant idea here is that we proposed only two verbal classes for all Arabic verbs.

Value Position 2	p-stem	i-stem	m-stem
L	FFVFWF	V-FFFWF	FFFWF
H	FFVFFWF	V-FFFFWF	FFFFWF

Apart from the extremely reduced categorization, the most valuable property of the proposed templates is the way they interact with the root and derivational material—the root and the derivational affixes are joined to form a discontinuous string—that we represent linearly. Then, each of the characters in the root plus affixes amalgam replaces each of the Fs in the template starting from the end. The Fs which are not replaced are simply removed from the resulting stem.

This joining algorithm has interesting advantages. First, it handles the apparent shifts of radicals along the template in an elegant way. This shift is encountered, for instance, when comparing patterns such as Iau and IX. Below we find the p-stems of the patterns I and IX, both of which follow template L '(F.)Fv.FwF'. The root consonants 'f', 'ç' and 'l' are located in different parts of the whole structure of the stems, but we do not have to indicate this in any way, the algorithm itself handles such operations.

pattern	p-stem	template
Iau	(F.)fa.çal	L
IX	f.ça.lal	L

Furthermore, another consequence of this categorization, which we already mentioned above, is that there are not different conjugational classes for verbs from triliteral and quadriliteral roots, both are treated exactly the same way—in this respect we move away from Cahill (2010), who handles triliteral and quadriliteral verbs

separately. For instance, both patterns II and QI stems are built on template H, i.e. F.FVF.FWF.

pattern	root type	p-stem	template
II	triliteral	faç.çal	H
QI	quadriliteral	faç.lab	H

It would be interesting to examine more carefully why all verbs seem to be divided in two axiomatic templates and why we could not reduc the system to a single class. In the following table we will show all the verbal patterns as if they followed both templates H and L. The yellow parts correspond to the 'nonexistent' template adjustments, i.e. the corresponding pattern does not follow the shaded template, but the unshaded one. We only represent p-stem and i-stem, as m-stem can be deduced from i-stem. There is just one representation of the six I patterns—as they only differ in the content of the second vowel of both stems.

Pattern	Template H (...)FvF.FvF		Template L (...)Fv.FvF	
	p-stem	i-stem	p-stem	i-stem
I	-	-	fa.çvl	af.çvl
II	faç.çal	u.faç.çil	*f.ça.çal	*uf.ça.çil
III	faA.çal	u.faA.çil	-	-
IV	caf.çal	uf.çil	*c.fa.çal	*u.fa.çil
V	ta.faç.çal	a.ta.faç.çal	*taf.ça.çil	*a.taf.ça.çil
VI	ta.faA.çal	a.ta.faA.çal	-	-
VII	*naf.çal	*a.naf.çil	n.fa.çal	an.fa.çil
VIII	*fat.çal	*a.fat.çil	f.ta.çal	af.ta.çil
IX	*faç.lal	*a.faç.lil	f.ça.lil	af.ça.lil
X	s.taf.çal	as.taf.çil	*sat.fa.çal	*a.sat.fa.çil
XI	f.çaA.lal	af.çaAlil	-	-
XII	f.çaw.çal	af.çaw.çil	*faç.wa.çal	*a.faç.wa.çil
XIII	f.çaw.wal	af.çaw.wil	*faç.wa.wal	*a.faç.wa.wil
XIV	f.çan.lal	af.çan.lil	*faç.na.lal	*a.faç.na.lil
XV	f.çan.laA	f.çan.liy	*faç.na.laA	*a.faç.na.liy
QI	faç.lab	u.faç.çib	*f.ça.lab	*uf.ça.lib
QII	ta.faç.lab	a.ta.faç.lib	*taf.ça.lab	*a.taf.ça.lib
QIII	f.çan.lab	af.çan.lib	*faç.na.lab	*a.faç.na.lib
QIV	f.çal.bab	af.çal.bib	*faç.la.bab	*a.faç.la.bib

A priori, it does not seem possible to extract consistent conclusions on the division of verbs in two classes, and why one type of verbs does not follow the other. We tentatively infer that each templatic class may have an etymological basis. Perhaps it would be interesting to examine the degree of coocurrency between each two verbal patterns to see which pairs are more related. We have pointed out that Danks (2011) carried out a quantitative analysis of these coocurrences in the verbal lexicon. A further step would be to compare both data.

With regards to the derivational material, we have delimited and listed the affixes present in the different patterns of the verbal system. We follow Danks (2011) in his definition of the derivational characteristic of Patterns III 'faAçal' as a vocalic lengthening process, instead of a vocalic addition. Also in Danks, the function of the prefix 'tV' of patterns V, VI and QII is described as a *detransitivizer*, which reduces the valency of the affected verb. This affix enjoys a special status in our system—as it is the only affix consisting of a full syllable, we add it to the verbal stem after having inserted the root plus affixation amalgam into the template. We decided to define it as a prefix –ta, and to have a special rule for changing a into u, in those cases where the prefix has the form –tu.

With regards to the vocalization exhibited by the different verbs, it is true that there seems to be some arbitrariness in the vowel qualities of each pattern, but we have seen that, for instance, there is a tendency to identify intransitive and stative meanings with 'u' and 'i' vowels in the p-stem. Yet, there is more interesting evidence to argue in favour of a well-justified vocalism morpheme—the formation of an internal passive exclusively by means of apophonic processes. The formation of active voice, however, seems to be more lexical.

One of the disadvantages of the generation system is that it tends to overgenerate. One of the reasons for this is that we used a large coverage lexicon of lemmas. However, the lexicon could be considerably reduced in case it is needed for a specific application. On the other hand, the passive voice is one of the most complex problems for overgeneration. The Arabic internal passive is not very productive in real usage, but its existence is generally determined by the semantics of the specific verb. As a result, the formation of the doubtful passives increases considerably the size of the inflected lexicon.

Another weakness of the analysis is that part of the Arabic lexicon seems not to follow the root-and-pattern or root-and-template approach. "(...) there are two aspects of the theory which remain problematic and require further revision: First the idea that all words are uniquely decomposable into root, patterns, and vowel melodies, and second the idea that the template is itself a morpheme." (Ratcliffe, 1998:26) In our categorization, we have said that the template is the result of phonotactic adjustments. Yet we defined two axiomatic structures which classify verbs in two morphological types—what we called templates L and H. Apart from this, we have analyzed all lexical items as a combination of roots, derivational material and vocalism, but there are indeed words which cannot be decomposed in these parts. We do not believe that this reality invalidates our analysis. It simply makes it necessary to have a separate analysis for *non reduceable* lexical items. Further, it is important to note that this type of words are not found in the lexicon of verbs, but it is a peculiarity of the nominal system. In fact, we have stated that the nominal system is far more idiosyncratic than that of the verb. We subscribe here to the words of Watson: "the canonical pattern of the Arabic verb is far more restricted than that of the noun"(Watson, 2007:133). Indecomposable nouns may be simply seen as another characteristic of a heterogeneous system.

As for the various computational resources developed by this project—the Jabalín morphological analyzer and generator, the Jabalín Online Interface, the lexicons of lemmas and inflected forms, and the transliteration scheme—are meant to be useful and valuable tools available for the research community. Moreover, the Jabalín Online Interface aims to have a pedagogical contribution too; it may serve as a helpful tool for second language learners.

To ensure the reliability of these tools we carried out an evaluation against the ElixirFm lexicon. The ElixirFm analyzer inherited all the lexicographic information available in the BAMA analyzer, and performed some interesting improvements. We believe it to be a robust morphological analyzer, and it has the advantage of being

freely distributed. We achieved a 99.52% precision on the evaluated forms, which is a very satisfactory result. In relation to the recall rates, we have stated that ElixirFM is not a reliable source for comparison. Yet, we have notieced that this is one of the main drawbacks of our lexicon of lemmas. As a future endeavour, we must measure recall using a corpus so that we can eliminate lexical entries no longer attested. To our knowledge, Attia et al. (2011b) are the only ones who have developed a representative lexicon of MSA.

The overall conclusion of the descriptive model is that all the derivational traits are harmonically combined in our model, which empathizes the subtle regularity and consistency encountered in the morphological system of Arabic. In this sense, we subscribe to the opinion of Danks: "the verbal morphology of MSA is highly systematic and lends itself to quantitative analysis which reveals non-random distributions of verbal patterns by root within the lexicon. Some of these distributions are attributable to morphological dependencies between patterns while others suggest that semantic or syntactic factors may be responsible" (Danks, 2011:37)

To this respect, we further add an observation made by Ratcliffe, who shares a similar position: "(…) it appears that each of these forms [IV, VII, VIII, X] contains a stable sequence ..CaCa (perf.) ..CiCu (imp.) and the material to the left of these sequences is automatically syllabified according to universal principles of syllabification. (…) We conclude, therefore, that these derived stems are not formed by mapping to a template, but simply by prefixation followed by predictable phonological readjustment" (Ratcliffe, 1998: 30).

The most remarkable conclusion we take from the template categorization and the ordering algorithm is that Arabic syllabic structure is overwhelmingly regular. The highly restrictive phonotactic system of Arabic makes the syllabic structure of stems predictable. In a nutshell, we have demonstrated that it is possible to develop a precise formalism which predicts the syllabic structures for Arabic lexical items.

We strongly believe that, in the long run, a morphological system based on a precise description of the Arabic morphological system would benefit from high efficiency and better adaptability to numerous applications. Therefore, our forthcoming endeavours will be focused on extending the proposed model to nominal morphology, so that we can develop a complete system to handle Arabic morphology.

The nominal system has the disadvantage of being more complex than verbal morphology, yet we believe that the basic principles of our analysis would be maintained in a nominal model.

We felt that any computational system which aims to emulate the morphology of a language can not be complete without semantic knowledge. Hence, in the near future, we find it essential to include semantic information in our model.

Conclusions

It has been proposed a linguistically consistent and complete model which accounts for all morpho-phonological and orthographic phenomena affecting the Arabic verbal system. Furthermore, a subsequent computational formalization of the model has been described and implemented focused on generation. The conclusions drawn from the objectives and the basic principles of the model are as follow:

1. We demonstrated that it is feasible to develop a model of Arabic verbal morphology based on a multi-level representation of stems in root, affixation, vocalism and template, and an adjustment algorithm for building the surface representation of the stem. Inflectional morphology is added to the stem by concatenative procedures. The surface representation of word-forms is handled by a set of phonotactic constraint rules, as well as rules for orthographic normalization. Some conclusions are drawn from the descriptive model:

    a. **The Arabic system of syllabic structure is overwhelmingly regular** and the possible prosodic patterns are so restricted that it can be described in a precise and simple formalization. The formalization used a moraic description.

    b. **Arabic verbs can be reduced to a system of two conjugational classes**. The difference between the two types is that, in one of them, the penultimate syllable of the stem is light, while in the other it is heavy.

    c. **Verbs built upon triliteral and quadriliteral roots can be described as belonging to the same conjugational class**.

d. **Pure irregular forms are widely incidental in Arabic verbs**. Alterations found in the surface representation of forms are the result of phonological constraints and orthographic conventions. Irregularity seems to be lexical, not morphological.

2. **Creation of a lexicon of lemmas and a lexicon of inflected forms** with linguistic feature specifications. The lexicon of lemmas has 15,452 entries. The lexicon of inflected forms includes 1,684,268 verbal forms.

3. **Evaluation of the lexicon of inflected forms with a precision of 99.52% accuracy**.

4. **Development of the Jabalín open-source morphological analyzer and generator and the Jabalín Online Interface** for sending queries to the morphological tools. Jabalín holds a GNU license.

Conclusiones (Spanish)

Hemos propuesto un modelo completo y lingüísticamente coherente que explica todos los fenómenos morfofonológicos y ortográficos del sistema verbal del árabe. El modelo verbal ha sido descrito formalmente e implementado desde el punto de vista de la generación. A partir de este modelo, y atendiendo a los objetivos que se propusieron al principio del trabajo, se han extraído las siguientes conclusiones:

1. Hemos demostrado la viabilidad de desarrollar un modelo de la morfología verbal del árabe basado en una representación multinivel del tema (*stem*) del verbo, dividido en las unidades léxicas: raíz, afijación, vocalismo y esquema (*template)*, por medio de un algoritmo *de ajuste* que construye la representación superficial del tema verbal. El paradigma flexivo se añade al tema a través de procedimientos concatenativos. La representación superficial de las formas verbales se deriva de la aplicación de un conjunto de reglas de restricciones fonotácticas, así como reglas para la normalización ortográfica. Se derivan varias conclusiones de este modelo descriptivo:

    a. **El sistema de estructuras silábicas del árabe es abrumadoramente regular** y los patrones prosódicos pueden ser descritos mediante una formalización precisa y simple. Esta formalización se basa en la unidad moraica para describir las posible estructuras.

    b. El sistema verbal árabe puede describirse a partir de la separación de dos únicas clases de conjugación. La diferencia entre estos dos tipos de verbos es que, en uno, la penúltima sílaba será ligera, mientras que en el otro, la penúltima sílaba será pesada.

c. **Los verbos construidos a partir de raíces trilíteras y cuadrilíteras pueden considerarse miembros de una misma clase conjugacional**, tal como han sido descritos en el modelo.

d. Las formas verbales puramente irregulares en árabe son anecdóticas. Las alteraciones encontradas en la representación superficial de las formas son el resultado de meras restricciones fonológicas y convenciones ortográficas. En un principio la irregularidad parece ser léxica en todos los casos, no morfológica.

2. **Creación de un lexicón de lemas verbales y un lexicón de formas flexionadas** anotados lingüísticamente. El lexicón de lemas cuenta con 15.452 entradas, mientras que el lexicón de formas flexionadas cuenta con 1,684,268 formas verbales.

3. **Evaluación del lexicón de formas flexionadas obteniendo una precisión del 99,52%** de las formas que han podido evaluarse.

4. **Desarrollo de una herramienta morfológica de código abierto para el análisis y la generación**, **el sistema Jabalín**, **así como la Interfaz Online Jabalín**. Jabalín se distribuye a través de una licencia GNU.

References

Abu-Chacra. (2007). *Arabic: An Essential Grammar*. New York: Taylor & Francis.

Al Shamsi, F., & Guessoum, A. (2006). A hidden Markov model-based POS tagger for Arabic. *Proceeding of the 8th International Conference on the Statistical Analysis of Textual Data, France* (pp. 31–42).

Al-Ani, S. (2007). The linguistic analysis and rules of pause in Arabic. In E. Ditters & H. Motzki (Eds.), (pp. 209–244).

Alderete, J., Tupper, P., & Frisch, S. A. (2010). Integrating connectionist and symbolic approaches to phonotactics: Arabic root cooccurrence restrictions revisited.

Alkuhlani, S., & Habash, N. (2011). A corpus for modeling morpho-syntactic agreement in Arabic: gender, number and rationality. Retrieved from http://www.aclweb.org/anthology-new/P/P11/P11-2062.pdf

Al-Najem, S. R. (2007). Inheritance-based approach to Arabic verbal root-and-pattern morphology. In Abdelhadi Soudi, A. van den Bosch, G. Neumann, & N. Ide (Eds.), (pp. 67–88). Retrieved from http://www.springerlink.com/content/q01213h567748811/abstract/

Al-Nassir, A. A. (1993). *Sibawayhi the phonologist*. Library of Arabic linguistics. New York: Kegan Paul International.

Al-Sughaiyer, I. A., & Al-Kharashi, I. A. (2004). Arabic morphological analysis techniques: a comprehensive survey. *J. Am. Soc. Inf. Sci. Technol.*, *55*(3), (pp. 189–213). doi:10.1002/asi.10368

Altantawy, M., Habash, N., & Rambow, O. (2011). Fast Yet Rich Morphological Analysis. *International Workshop Finite State Methods and Natural Language Processing* (p. 116). Retrieved from http://aclweb.org/anthology-new/W/W11/W11-44.pdf#page=128

Altantawy, M., Habash, N., Rambow, O., & Saleh, I. (2010). Morphological analysis and generation of Arabic nouns: A morphemic functional approach. *Proceedings of the seventh International Conference on Language Resources and Evaluation (LREC), Valletta, Malta.*

Andersen, G. (2010). Applying corpus linguistics to other areas of research. In A. O'Keeffe & M. McCarthy (Eds.), *The Routledge Handbook of Corpus Linguistics* (1st ed., pp. 546–562). Routledge.

Attia, M. A. (2005). Developing Robust Arabic Morphological Transducer Using Finite State Technology. Paper presented at The. *8th Annual CLUK Research Colloquium*.

Attia, M., Pecina, P., Toral, A., Tounsi, L., & van Genabith, J. (2011a). An Open-Source Finite State Morphological Transducer for Modern Standard Arabic. *International Workshop Finite State Methods and Natural Language Processing* (p. 125). Retrieved from http://acl.eldoc.ub.rug.nl/mirror/W/W11/W11-44.pdf#page=137

Attia, Mohammed, Pecina, P., Toral, A., Tounsi, L., & Genabith, J. van. (2011b). A lexical database for Modern Standard Arabic interoperable with a finite state morphological transducer. *Communications in Computer and Information Science*, *100*, (pp. 89–118).

Badawi, E.-S. M., Carter, M. G., & Gully, A. (2004). *Modern Written Arabic: A Comprehensive Grammar*. London: Routledge.

Bahloul, M. (2008). *Structure and function of the Arabic verb*. Routledge.

Bassiouney, R. (2009). *Arabic Sociolinguistics*. Edinburgh: Edinburgh University Press.

Beesley, K. R. (1998c). Arabic morphological analysis on the Internet. *Proceedings of the 6th International Conference and Exhibition on Multi-lingual Computing*.

Beesley, K. R. (2001). Finite-state morphological analysis and generation of Arabic at Xerox Research: Status and plans in 2001. *ACL Workshop on Arabic Language Processing: Status and Perspective* (Vol. 1, pp. 1–8). Retrieved from http://www.xrce.xerox.com/content/download/20547/147632/file/finite-state.pdf

Beesley, K.R. (1998b). Arabic morphology using only finite-state operations. *Proceedings of the Workshop on Computational Approaches to Semitic languages* (pp. 50–7).

Beesley, Kenneth R. (1999). Arabic stem morphotactics via finite-state intersection. In E. Benmamoun (Ed.), (pp. 85–100).

Benmamoun, E. (Ed.). (1999). *Perspectives on Arabic Linguistics XII: Papers from the Twelfth Annual Symposium on Arabic Linguistics*. Current issues in linguistic theory. Amsterdam: John Benjamins Publishing Company.

References

Berent, I., & Everett, D. L. (2001). Do phonological representations specify variables? Evidence from the Obligatory Contour Principle. *Cognitive Psychology*, *42*(1), (pp. 1–60).

Bird, S., & Blackburn, P. (1991). A logical approach to Arabic phonology. *Proceedings of the fifth conference on European chapter of the Association for Computational Linguistics* (pp. 89–94). Retrieved from http://dl.acm.org/citation.cfm?id=977196

Bohas, George. (2006). The organization of the lexicon in Arabic and other Semitic languages. In Sami Boudelaa (Ed.), . Retrieved from http://webs.ono.com/iferrando/alscambridgePP.pdf

Bohas, Georges, & Saguer, A. (2007). The explanation of homonymy in the lexicon of Arabic. In E. Ditters & H. Motzki (Eds.), (pp. 255–290).

Boudelaa, S., & Marslen-Wilson, W. D. (2001). Morphological units in the Arabic mental lexicon. *Cognition*, *81*(1), (pp. 65–92).

Boudelaa, Sami (Ed.). (2006). *Perspectives on Arabic Linguistics XVI: Papers from the Sixteenth Annual Symposium on Arabic Linguistics, Cambridge, March 2002*. Amsterdam: John Benjamins Publishing.

Brame, M. (1970). *Arabic phonology: implications for phonological theory and historic Semitic*. Massachusetts Institute of Technology.

Buckwalter, T. (2004). *Buckwalter Arabic Morphological Analyzer (BAMA) Version 2.0*. Linguistic Data Consortium (LDC).

Buckwalter, T., & Parkinson, D. (2010). *A Frequency Dictionary of Arabic: Core Vocabulary for Learners* (Bilingual.). London; New York: Routledge.

Cahill, L. (2010). A Syllable-based approach to verbal morphology in Arabic. *Editors & Workshop Chairs* (p. 19).

Cahill, L. J. (1990). Syllable-based morphology. *Proceedings of the 13th conference on Computational linguistics-Volume 3* (pp. 48–53).

Cahill, Lynne. (2007). A Syllable-based Account of Arabic Morphology. In Abdelhadi Soudi, A. van den Bosch, G. Neumann, & N. Ide (Eds.), (pp. 45–66). Retrieved from http://www.springerlink.com/content/q01213h567748811/abstract/

Cavalli-Sforza, V., Soudi, A., & Mitamura, T. (2000). Arabic morphology generation using a concatenative strategy. *Proceedings of the 1st North American chapter of the Association for Computational Linguistics conference* (pp. 86–93). Retrieved from http://dl.acm.org/citation.cfm?id=974317

Comrie, B., & Kaye, A. S. (Eds.). (1990). Arabic. *The World's major languages* (pp. 664–685). Oxford University Press.

Cooper, R., & Ranta, A. (2008). Natural languages as collections of resources. *Language in Flux: Relating Dialogue Coordination to Language Variation, Change and Evolution. College Publications, London.* Retrieved from http://edu.cs.uni-magdeburg.de/EC/lehre/wintersemester-2010-2011/seminar-ein-grammatikformalismus-fuer-multilinguale-sprach-und-dialogverarbeitung/informationen-zum-seminar/files/Natural%20languages%20as%20collections%20of%20resources.pdf

Corriente, F. (1970). *Diccionario español-árabe.* Madrid: Instituto Hispano-Árabe de Cultura.

Corriente, F. (1991). *Diccionario árabe-español.* Madrid: Editorial Herder.

Corriente, F. (2004). Geminate imperfectives in Arabic masked as intensive stems of the verb. *Estudios de dialectología norteafricana y andalusí, 8*, 33–57.

Cortés, J. (1996). *Diccionario de Arabe culto moderno.* Madrid: Rba Publicaciones Editores revistas.

Cowan, D. (1958). *An Introduction to Modern Literary Arabic.* Cambridge: Cambridge University Press.

Dakkak, O. A., Ghneim, N., Alshalaby, A., Sonbol, R., & Desouki, M. S. (2009). Arabic Language Resources in HIAS. Presented at the Proceedings of the Second International Conference on Arabic Language Resources and Tools. Retrieved from http://www.elda.org/medar-conference/pdf/50.pdf

Danks, W. (2011). *The Arabic Verb: Form and Meaning in the Vowel-Lengthening Patterns.* Amsterdam: John Benjamins Publishing Company.

Dichy, J. (2000). Morphosyntactic Specifiers to be associated to Arabic Lexical Entries-Methodological and Theoretical Aspects. *Proceedings of the ACIDA'2000 conference, Monastir (Tunisia)* (pp. 22–24).

Dichy, J. (2002). Arabic lexica in a cross-lingual perspective. *Proceedings of ARABIC Language Resources and Evaluation: Status and Prospects, A Post Workshop of LREC.*

Dichy, Joseph. (2007). Fa'ula, fa'ila, fa'ala: dispersion et régularités sémantiques dans les trois schèmes simples du verbe arabe. In E. Ditters & H. Motzki (Eds.), (pp. 313–366). Leiden: Brill.

References

Ditters, E., & Motzki, H. (2007). *Approaches to Arabic Linguistics: Presented to Kees Versteegh on the Occasion of His Sixtieth Birthday*. Leiden: Brill.

Eid, M., & McCarthy, J. (Eds.). (1990). *Perspectives on Arabic linguistics II: papers from the Second Annual Symposium on Arabic Linguistics*. Amsterdam: John Benjamins Publishing Company.

El Kholy, A., & Habash, N. (2011). Automatic Error Analysis for Morphologically Rich Languages. *Proceedings of the Thirteen Machine Translation summit (MT Summit XIII)* (pp. 225–232). Xiamen, China. Retrieved from http://www.mt-archive.info/MTS-2011-ElKholy.pdf

El-Beltagy, S. R., & Rafea, A. (2009). A framework for the rapid development of list based domain specific Arabic stemmers. *Proceedings of the 2nd International Conference on Arabic Language Resources and Tools*.

El-Dahdah, A. (1991). *A Dictionary of Arabic Verb Conjugation*. Beirut: Librairie du Liban.

Erwin, W. M. (1963). *A Short Reference Grammar of Iraqi Arabic*. Washington, D.C.: Georgetown University Press.

Ethnologue report for language code: arb. (n.d.). Retrieved April 27, 2012, from http://www.ethnologue.com/show_language.asp?code=arb

Farghaly, A. (2012). Statistical and symbolic paradigms in Arabic computational linguistics. In R. Bassiouney & G. Katz (Eds.), *Arabic Language and Linguistics*. Georgetown University Press.

Ferrando, I. (1999). El plural fracto en semítico: nuevas perspectivas. *Estudios de dialectología norteafricana y andalusí, EDNA*, (4), (pp. 7–24).

Ferrando, I. (2006). The plural of paucity in Arabic and its actual scope. On two claims by Siibawayhi and al-Farraa'. In Sami Boudelaa (Ed.), . Retrieved from http://webs.ono.com/iferrando/alscambridgePP.pdf

Frisch, S. A., Pierrehumbert, J. B., & Broe, M. B. (2004). Similarity avoidance and the OCP. *Natural Language & Linguistic Theory*, *22*(1), (pp. 179–228).

Gabbard, K. M. (2010). *A Phonological Analysis of Somali And the Guttural Consonants*. The Ohio State University. Retrieved from https://kb.osu.edu/dspace/bitstream/handle/1811/46639/gabbardsomalithesis2010.pdf?sequence=1

Geers, F. W. (1945). The treatment of emphatics in Akkadian. *Journal of Near Eastern Studies*, *4*(2), 65–67.

González Martínez, A. (2010). *Un modelo de analizador morfológico automático de verbos en árabe*. Universidad Autónoma de Madrid, Madrid.

Goweder, A., & De Roeck, A. (2001). Assessment of a significant Arabic corpus. *Arabic NLP Workshop at ACL/EACL*.

Gridach, M., & Chenfour, N. (2011). Developing a New System for Arabic Morphological Analysis and Generation. *Proceedings of the 2nd Workshop on South and Southeast Asian Natural Language Processing (WSSANLP 2011)* (p. 52). Retrieved from http://igm.univ-mlv.fr/~voyatzi/WS1_WSSANLP/WSSANLP-2011.pdf#page=66

Gussenhoven, C., & Jacobs, H. (1998). *Understanding Phonology*. London: Arnold.

Habash, N., & Rambow, O. (2006). MAGEAD: a morphological analyzer and generator for the Arabic dialects. *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics* (pp. 681–688). Retrieved from http://dl.acm.org/citation.cfm?id=1220261

Habash, N., Rambow, O., & Kiraz, G. (2005). Morphological analysis and generation for Arabic dialects. *Proceedings of the ACL Workshop on Computational Approaches to Semitic Languages* (pp. 17–24). Retrieved from http://acl.ldc.upenn.edu/W/W05/W05-07.pdf#page=27

Habash, N., Rambow, O., & Roth, R. (2009). Mada+tokan: A toolkit for arabic tokenization, diacritization, morphological disambiguation, pos tagging, stemming and lemmatization. *Proceedings of the 2nd International Conference on Arabic Language Resources and Tools (MEDAR), Cairo, Egypt* (pp. 242–245).

Habash, N. Y. (2010). *Introduction to Arabic Natural Language Processing*. UK: Morgan & Claypool Publishers.

Habash, Nizar, Soudi, A., & Buckwalter, T. (2007). On Arabic Transliteration. In Abdelhadi Soudi, A. van den Bosch, & Günter Neumann (Eds.), (pp. 15–22). Retrieved from http://nizarhabash.com/publications/chapter2BisHabash_et_al-2007-web.pdf

References

Hallman, P. (2006). Causativity and transitivity in Arabic. Retrieved from http://web.mit.edu/ammar/24901/Causativity.pdf

Haspelmath, M., & Sims, A. D. (2010). *Understanding Morphology*. Oxford: Hodder Education.

Hassanein, A. (2006). *Modern Standard Arabic Grammar: A Concise Guide*. Cairo: American Univ in Cairo Press.

Hausser, R. R. (2001). *Foundations of Computational Linguistics: Human-Computer Communication in Natural Language*. New York: Springer.

Haywood, J. A., & Nahmad, H. M. (1962). *A new Arabic grammar of the written language*. London: Lund, Humphries.

Holes, C. (2004). *Modern Arabic: Structures, Functions, and Varieties*. Washington, D.C.: Georgetown University Press.

Idrissi, A., & Kehayia, E. (2004). Morphological units in the Arabic mental lexicon: Evidence from an individual with deep dyslexia. *Brain and language*, *90*(1-3), (pp. 183–197).

International Phonetic Association. (1999). *Handbook of the International Phonetic Association: a guide to the use of the International Phonetic Alphabet*. Cambridge: Cambridge University Press.

Janet Pierrehumbert. (1993). Dissimilarity in the Arabic verbal roots. *Proceedings of the Northeast Linguistic Society*. Retrieved from http://faculty.wcas.northwestern.edu/~jbp/publications/arabic_roots.pdf

John Goldsmith. (1979). The aims of autosegmental phonology. *Current Approach in Phonological theory*. Retrieved from http://hum.uchicago.edu/~jagoldsm/Papers/AimsAutosegmental.pdf

John McCarthy. (1990). Foot and word in prosodic morphology: The Arabic broken plural. Retrieved from http://works.bepress.com/cgi/viewcontent.cgi?article=1015&context=john_j_mccarthy

Joshi, A. K. (1987). Unification and some new grammatical formalisms. *Proceedings of the 1987 workshop on Theoretical issues in natural language processing* (pp. 45–50).

Jurafsky, D., & Martin, J. H. (2009). *Speech and language processing: an introduction to natural language processing, computational linguistics, and speech recognition*. United States: Prentice Hall.

Kanaan, G., Al-Shalabi, R., & Sawalha, M. (2003). Full automatic Arabic text tagging system. *the proceedings of the International Conference on Information Technology and Natural Sciences, Amman/Jordan* (pp. 258–267).

Karttunen, L., & Beesley, K. R. (2005). Twenty-five years of finite-state morphology. *Inquiries Into Words, a Festschrift for Kimmo Koskenniemi on his 60th Birthday*, (pp. 71–83).

Kay, M. (1987). Nonconcatenative finite-state morphology. *Proceedings of the third conference on European chapter of the Association for Computational Linguistics* (pp. 2–10). Retrieved from http://dl.acm.org/citation.cfm?id=976860

Kenneth R. Beesley. (1998a). Consonant spreading in Arabic stems. *COLING '98 Proceedings of the 17th international conference on Computational linguistics*, *1*, (pp. 117–123).

Khashan Mohammad Khashan 2003) .خشان محمد خشان). Al-Khalil and numerical prosody (1) الخليل بن أحمد والعروض الرقمي. *Journal of Arabic Linguistic Tradition*, *1*, (pp. 25–34).

Khoja, S. (2001). APT: Arabic part-of-speech tagger. *Proceedings of the Student Workshop at NAACL* (pp. 20–25).

Khoja, S., Garside, R., & Knowles, G. (2003). A tagset for the morphosyntactic tagging of Arabic. *Proceedings of the Corpus Linguistics. Lancaster University (UK)*, *13*.

Khoja, Shereen, & Garside, R. (1999). Stemming Arabic text. *Technical report, Computer Department, Lancaster University*.

Kiraz, G. A. (1996). Computing prosodic morphology. *Proceedings of the 16th conference on Computational linguistics-Volume 2* (pp. 664–669).

Kiraz, G. A. (1994b). Computational analyses of Arabic morphology. *Arxiv preprint cmp-lg/9408002*. Retrieved from http://arxiv.org/abs/cmp-lg/9408002

Kiraz, G. A. (2000). Multitiered nonlinear morphology using multitape finite automata: a case study on Syriac and Arabic. *Computational Linguistics*, *26*(1), (pp. 77–105).

Kiraz, George Anton. (1994a). Multi-tape Two-level morphology: a case study in Semitic non-linear morphology. Retrieved from http://acl.ldc.upenn.edu/C/C94/C94-1029.pdf

References

Kiraz, George Anton. (1999). Computational tool for developing morphophonological models for Arabic. In E. Benmamoun (Ed.), (pp. 101–112).

Kiraz, George Anton. (2001). *Computational Nonlinear Morphology: With Emphasis on Semitic Languages*. Cambridge: Cambridge University Press.

Kouloughli, D. E. (1986). Sur la structure interne des syllabes «lourdes» en arabe classique. *Revue québécoise de linguistique*, *16*(1).

Kulick, S., Bies, A., & Maamouri, M. (2010). Consistent and flexible integration of morphological annotation in the Arabic Treebank. *Language Resources and Evaluation (LREC)*.

Ladefoged, P., & Johnson, K. (2001). *A Course in Phonetics*. Boston: Cengage Learning.

Lieber, R. (2009). *Introducing Morphology*. Cambridge: Cambridge University Press.

Mace, J. (2007). *Arabic Verbs*. United States: Hippocrene Books.

Mas'ūd, A. I. 'Alī I., & Åkesson, J. (2001). *Arabic Morphology and Phonology: Based on the Marāḥ Al-arwāḥ by Aḥmad B. 'Alī B. Mas'ūd*. Boston: Brill.

McCarthy, J. J. (1981). A prosodic theory of nonconcatenative morphology. *Linguistic inquiry*, *12*, (pp. 373–418).

McCarthy, J. J. (1993). Template form in prosodic morphology.

McCarthy, J. J. (1994). The Phonetics and Phonology of Semitic Pharyngeals, *in P. Keating (ed., 1994), Papers in Laboratory Phonology III, Cambridge University Press, Cambridge*, (pp. 191–233).

McCarthy, J. J., & Prince, A. (1986). Prosodic morphology 1986.

McCarthy, J., & Prince, A. (1990). Prosodic morphology and templatic morphology. In M. Eid & J. J. McCarthy (Eds.), (pp. 1–54).

Mitkov, R., & Trost, H. (Eds.). (2005). Morphology. *The Oxford Handbook Of Computational Linguistics* (pp. 25–47). Oxford University Press.

Moreno Sandoval, A. (1993). *Un modelo computacional basado en la unificación para el análisis y generación de la morfología del español*. Universidad Autónoma de Madrid. Retrieved from http://dialnet.unirioja.es/servlet/libro?codigo=177873

Moreno Sandoval, A. (2001). *Gramáticas de unificación y rasgos*. Madrid: A. Machado Libros.

Moreno Sandoval, A. (2009). Panorama actual de la ingeniería lingüística. In A. Alcina & E. Valero (Eds.), *Terminología y Sociedad del Conocimiento* (pp. 99–116). Peter Lang.

Moreno-Sandoval, A., & Goñi-Menoyo, J. M. (2002). Spanish Inflectional Morphology in DATR. *Journal of Logic, Language and Information, 11*(1), 79–105. doi:10.1023/A:1013019622647

Musael S. Bin-Muqbil. (2006). *Phonetic and phonological aspects of Arabic emphatics and gutturals (dissertation)*. Retrieved from http://faculty.ksu.edu.sa/bin-muqbil/Publications/Bin-Muqbil_2006_Dissertation.pdf

Neme, A. A. (2011). A lexicon of Arabic verbs constructed on the basis of Semitic taxonomy and using finite-state transducers. *First International Workshop on Lexical Resources* (p. 78).

Newman, D. (2002). The phonetic status of Arabic within the world's languages: the uniqueness of the lughat al-daad. *Antwerp papers in linguistics., 100*, (pp. 65–75).

Nizar Habash, & Owen Rambow. (2007). Arabic Diacritization through Full Morphologial Tagging. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=7903DDA52827A48A2061842DF8A016EA?doi=10.1.1.101.6049&rep=rep1&type=pdf

O'Grady, W. D., Dobrovolsky, M., & Katamba, F. (1996). *Contemporary linguistics: an introduction*. Essex: Longman.

Ouersighni, R. (2001). A major offshoot of the DIINAR-MBC project: AraParse, a morphosyntactic analyzer for unvowelled Arabic texts. *ACL 39th Annual Meeting* (pp. 9–16).

Owens, J. (2006). *A Linguistic History of Arabic*. Oxford: Oxford University Press, USA.

Padgett, J. (2002). Feature Classes in Phonology. *Language, 78.2*.

Rashwan, M., Al-Badrashiny, M., Attia, M., & Abdou, S. (2009). A hybrid system for automatic arabic diacritization. *The 2nd International Conference on Arabic Language Resources and Tools*.

Ratcliffe, R. R. (1998). *The "broken" plural problem in Arabic and comparative Semitic: allomorphy and analogy in non-concatenative morphology*. Amsterdam: John Benjamins Publishing Company.

Reig, D. (1983). *La conjugaison arabe - كتاب التصريف*. Paris: Maisonneuve & Larose.

References

Roark, B., & Sproat, R. W. (2007). *Computational approaches to morphology and syntax*. United States: Oxford University Press.

Roca, I., & Johnson, W. (1999). *A Course in Phonology*. Oxford: Wiley.

Rodrigues, P., & Cavar, D. (2007). Learning Arabic Morphology Using Statistical Constraint-Satisfaction Models. In E. Benmamoun (Ed.), *Papers from the nineteenth Annual Symposium on Arabic Linguistics, Urbana, Illinois, april 2005* (pp. 63–76). John Benjamins Publishing.

Roth, R., Rambow, O., Habash, N., Diab, M., & Rudin, C. (2008). Arabic morphological tagging, diacritization, and lemmatization using lexeme models and feature ranking. *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics on Human Language Technologies: Short Papers* (pp. 117–120).

Ryding, K. C. (2005). *A reference grammar of modern standard Arabic*. Cambridge: Cambridge University Press.

Sag, I. A., Kaplan, R., Karttunen, L., Kay, M., Pollard, C., Shieber, S., & Zaenen, A. (1986). Unification and grammatical theory. Retrieved from http://lingo.stanford.edu/sag/papers/sagetal-1986.pdf

Samy, D. (2005). *Recursos bilingües de ingeniería lingüística para el procesamiento de español y árabe.* Universidad Autónoma de Madrid.

Sawalha, M., & Atwell, E. S. (2008). Comparative evaluation of arabic language morphological analysers and stemmers. *Proceedings of COLING 2008 22nd International Conference on Comptational Linguistics (Poster Volume))* (pp. 107–110). Retrieved from http://eprints.whiterose.ac.uk/42635/

Shalaan, K., Samy, D., & Magdi, M. (2010). Towards Resolving Morphological Ambiguity in Arabic Intelligent Language Tutoring Framework. *Proceedings of the first Workshop on Supporting eLearning with Language Resources and Semantic Data* (pp. 12–17). Presented at the LREC workshop 22 May 2010.

Shimron, J. (2003). *Language Processing and Acquisition in Languages of Semitic, Root-Based, Morphology*. Amsterdam: John Benjamins Publishing Company.

Smrž, O. (2007a). ElixirFM: implementation of functional Arabic morphology. *Proceedings of the 2007 Workshop on Computational Approaches to Semitic Languages: Common Issues and Resources* (pp. 1–8).

Smrž, Otakar. (2007b). *Functional Arabic morphology. Formal system and implementation*. Retrieved from http://ufal.mff.cuni.cz/~smrz/elixir-thesis.pdf

Smrž, Otakar, Bielický, V., Kouřilová, I., Kráčmar, J., Hajič, J., & Zemánek, P. (2008). Prague Arabic Dependency Treebank: a word on the million words. Retrieved from http://ufal.mff.cuni.cz/~smrz/LREC2008/padt-lrec.pdf

Souag, L. (2002). Broken plurals - or infixes? The case of the Algerian Arabic Dellys. *Estudios de dialectología norteafricana y andalusí*, *6*, (pp. 19–34).

Soudi, A., Cavalli-Sforza, V., & Jamari, A. (2001). A Computational Lexeme-Based Treatment of Arabic Morphology. *Proceedings of the Arabic Natural Language Processing Workshop, Conference of the Association for Computational Linguistics (ACL 2001)* (pp. 50–57).

Soudi, A., & Eisele, A. (2004). Generating an Arabic full-form lexicon for bidirectional morphology lookup. *Proceedings of LREC*. Retrieved from http://www.lrec-conf.org/proceedings/lrec2004/pdf/567.pdf

Soudi, Abdelhadi, Bosch, A. van den, & Neumann, G. (2007a). Arabic Computational Morphology: Knowledge-based and Empirical Methods. In Abdelhadi Soudi, A. van den Bosch, & G. Neumann (Eds.), (pp. 3–14). Springer.

Soudi, Abdelhadi, Bosch, A. van den, & Neumann, G. (Eds.). (2007b). *Arabic Computational Morphology: Knowledge-based and Empirical Methods*. Text, speech and language technology (Vol. 38). The Netherlands: Springer.

Spencer, A. (1991). *Morphological Theory: An Introduction to Word Structure in Generative Grammar*. Oxford: Basil Blackwell.

Tucker, M. A. (2009). The Root–and–Prosody Approach to Arabic Verbs. *Ms., University of California, Santa Cruz*.

Versteegh, K. (1997). *Landmarks in linguistic thought III: the Arabic linguistic tradition*. London; New York: Routledge.

References

Versteegh, K. (2001). *The Arabic language*. Edinburgh: Edinburgh University Press.

Versteegh, K. (2008). *Encyclopedia of Arabic language and linguistics*. Leiden: Brill.

Watson, J. C. E. (2007). *The Phonology and Morphology of Arabic*. Oxford; New York: Oxford University Press, USA.

Wehr, H. (1993). *Arabic-English Dictionary: The Hans Wehr Dictionary of Modern Written Arabic*. (J. M. Cowan, Ed.) (4th ed.). United States: Spoken Language Services.

Wright, W. (2007). *A Grammar of The Arabic Language (Wright's Grammar). Vol-1 & Vol-2 Combined together (Third Edition)*. (3rd ed.). United States: Simon Wallenburg Press.

Zawaydeh, B. A., & Davis, S. (1999). Hypocoristic formation in Ammani-Jordanian Arabic. In E. Benmamoun (Ed.), .

ذات السلاسل، .*النحو الأساسي* .(محمد حماسة عبد اللطيف, & 1994) .احمد مختار عمر, مصطفى النحاس زهران

الباحث العربي: قاموس عربي عربي. (n.d.). Retrieved April 19, 2012, from http://www.baheth.info/

دار المشرق، .*المنجد في اللغة والاعلام* (1973).

مكتبة لبنان .*معجم قواعد اللغة العربية فى جداول ولوحات* .(انطوان الدحداح. (2008

دار الفكر المعاصر للطباعة والنشر والتوزيع .*كيف نتعلم الإعراب* .(توفيق بن عمر بلطهجي. (1997

دار الفكر للطباعة والنشر والتوزيع، .*القاموس المنجد: عربي - عربي* .(يوسف بقاعي. (2005 ,شهاب الدين أبو عمرو

دار النفائس للنشر والتوزيع .*المعجم الوسيط في الاعراب* .(نايف معروف. (2000

Appendixes

Appendix A: Transliteration system

The transliteration scheme is partially based on the Buckwalter transliteration proposed in (2002)[85] and, less directly, a revised version made by Habash, Soudi and Buckwalter (Habash et al., 2007). In the following table, the three transliteration conventions are compared.

Transliteration schemes				Example[86]			
Arabic	Jabalín	Buck-walter	Habash et al.	Arabic	Jabalín	Buck-walter	Habash et al.
ء	c	'	'	سَمَاء	samaAc	samaA'	samaA'
آ	Ã	|	Ā	آمَنَ	Ãmana	|mana	Āmana
أ	Á	>	Ǻ	سَأَلَ	saÁala	sa>ala	saǺala
ؤ	ú	&	ŵ	مُؤْتَمَر	muú·tamar	mu&otamar	muŵ.tamar
إ	À	<	Ǻ	إِنْتَرْنِت	Àin·tar·nit	<inotaronit	Ǻin.tar.nit
ئ	ý	}	ŷ	سَائِل	saAý·il	saA}il	saAŷil
ا	A	A	A	كَانَ	kaAna	kaAna	kaAna
ب	b	b	b	بَرِيد	bariyd	bariyd	bariyd
ة	ä	p	ħ	مَكْتَبَةٌ	mak·tabaäû	makotabapN	mak.tabaħũ
ت	t	t	t	تَنَافُس	tanaAfus	tanaAfus	tanaAfus
ث	þ	v	θ	ثَلَاثَة	þalaAþaä	valaAvap	θalaAθaħ
ج	j	j	j	جَمِيل	jamiyl	jamiyl	jamiyl
ح	H	H	H	حَادّ	HaAd~	HaAd~	HaAd~
خ	x	x	x	خُوذَة	xuwðaä	xuw*ap	xuwðaħ

[85] http://www.qamus.org/transliteration.htm

[86] The examples used to compare the three transliterations are taken from Habash and Rambow (2007).

د	d	d	d	دَلِيل	daliyl	daliyl	daliyl
ذ	ð	*	ð	ذَهَب	ðahab	*ahab	ðahab
ر	r	r	r	رَفِيع	rafiyç	rafiyE	rafiyç
ز	z	z	z	زِينَة	ziynaä	ziynap	ziynaħ
س	s	s	s	سَمَاء	samaAc	samaA'	samaA'
ش	X	$	š	شَرِيف	Xariyf	$ariyf	šariyf
ص	S	S	S	صَوْت	Saw·t	Sawot	Saw.t
ض	D	D	D	ضَرِير	Dariyr	Dariyr	Dariyr
ط	T	T	T	طَوِيل	Tawiyl	Tawiyl	Tawiyl
ظ	Z	Z	Ď	ظُلْم	Zul·m	Zulom	Ďul.m
ع	ç	E	ç	عَمَل	çamal	Eamal	çamal
غ	g	g	ɣ	غَرِيب	gariyb	gariyb	ɣariyb
ف	f	f	f	فِيلم	fiylm	fiylm	fiylm
ق	q	q	q	قَادِر	qaAdir	qaAdir	qaAdir
ك	k	k	k	كَرِيم	kariym	kariym	kariym
ل	l	l	l	لَذِيذ	laðiyð	la*iy*	laðiyð
م	m	m	m	مُدِير	mudiyr	mudiyr	mudiyr
ن	n	n	n	نُور	nuwr	nuwr	nuwr
ه	h	h	h	هَوْل	haw·l	hawol	haw.l
و	w	w	w	وَصَل	waSal	waSal	waSal
ى	Y	Y	ý	عَلَى	çalaY	EalaY	çalaý
ي	y	y	y	تِين	tiyn	tiyn	tiyn
◌َ	a	a	a	دَهَنَ	dahana	dahana	dahana
◌ُ	u	u	u	دُهِنَ	duhina	duhina	duhina
◌ِ	i	i	i	دُهِنَ	duhina	duhina	duhina
◌ً	â	F	ã	كِتَابًا	kitaAbâA	kitaAbFA	kitaAbãA
◌ٌ	û	N	ũ	كِتَابٌ	kitaAbû	kitaAbN	kitaAbũ
◌ٍ	î	K	ĩ	كِتَابٍ	kitaAbî	kitaAbK	kitaAbĩ
◌ّ	~	~	~	كَسَّرَ	kas~ar	kas~ar	kas~ar
◌ْ	·	o	.	مَسْجِد	mas·jid	masojid	mas.jid
ـ	¯	_	_	مَسْـجد	mas·¯jid	maso_jid	mas._jid

TABLE 29 Comparative table of Jabalín, Buckwalter and Habash et al. transliterations

To develop our transliteration, we adopted three criteria:

1. We restricted the election of characters to the list of letters and symbols provided by the ISO-8859-1 codification standard, commonly known as Latin 1, since this standard can be handled by the majority of platforms and is the default character set in most browsers.
2. We avoided the use of punctuation marks, for the transliteration may be used to represent a text, in which case the symbols would cause an irresolvable ambiguity together with punctuation.
3. We attempted to develop a readable mapping of Arabic letters, thus similarity and consistency between character correspondences is intended. The character selection is nevertheless restricted to Latin 1, so this criterion is subjected to character availability.

These three criteria used for developing the transliteration scheme ultimately aim for ease of use. Hence, the differences from the Buckwalter and Habash et al. transliterations are meant to be based on this principle—if the symbols proposed by those transliterations did not respect our criteria enough, or there were more suitable options in the Latin 1 inventory, we chose another character. In general, Buckwalter transliteration (2002) is focused on portability and consistency with computing encodings, for it restricted the character set to ASCII[87]. On the contrary, Habash et al. emphasised the importance of having a readable mapping. Therefore, Buckwalter and Habash et al. just give more weight either to portability (point 1) or readability (points 2 and 3).

For instance, for the letter ع *'ayn* Habash et al. (2007 version) uses the notation 'ç', and the letter غ the notation 'ɣ'. Conversely, we preferred to use ç' for the first one and 'g' for the second one, both available in Latin 1 and easier to write. Further, we

[87] American Standard Code for Information Interchange.

believe that their shape is quite straightforward to associate to the original Arabic letter. One difficult symbol to transliterate is the *sukun* diacritic '◌̥'. The Buckwalter transliteration (2002) uses the letter 'o', which presents the advantage of having a similar shape, but on the other hand, it may be confused with a vowel—this is a serious drawback since the sukun indeed indicates the absence of a vowel. Habash et al. proposes to represent the sukun as a dot '.'. This again violates our criteria in a decisive way, for it uses a punctuation mark, making the transliteration invalid to represent a text. Our choice was the interpunct '·', which does not infringe any of the criteria.

The ISO-8859-1 or Latin 1[88] is a codification standard which aims to represent mostly the alphabetic characters and symbols of the Western European languages. It is widely used and reasonably portable to any platform. The Latin 1 is a superset of the ASCII encoding[89], a seven-bit code which was primarily intended for the English language. Conversely, the Latin 1 makes use of eight bits, so it includes the 128 characters of the ASCII—generally English letters, digits, punctuation marks and other special characters, apart from a set of control codes—and adds another 128 characters representing mainly Latin letters with diacritics—and again some control codes. The Latin 1 is in turn a subset of the Unicode standard, i.e. the first 256 characters of Unicode correspond to the Latin 1 character set.

[88] http://en.wikipedia.org/wiki/ISO/IEC_8859-1

[89] http://www.unicode.org/charts/PDF/U0000.pdf

Arabic		Transliteration		
Letter	Codification	Letter	Description	Codification
ء	0x621	c	small c	0x63
آ	0x622	Ã	capital a-tilde	0xc3
أ	0x623	Á	capital a-acute	0xc1
ؤ	0x624	ú	small u-acute	0xfa
إ	0x625	À	capital a-grave	0xc0
ئ	0x626	ý	small y-acute	0xfd
ا	0x627	A	capital a	0x41
ب	0x628	b	small b	0x62
ة	0x629	ä	small a-diaeresis	0xe4
ت	0x62a	t	small t	0x74
ث	0x62b	þ	small thorn	0xfe
ج	0x62c	j	small j	0x6a
ح	0x62d	H	capital h	0x48
خ	0x62e	x	small x	0x78
د	0x62f	d	small d	0x64
ذ	0x630	ð	small eth	0xf0
ر	0x631	r	small r	0x72
ز	0x632	z	small z	0x7a
س	0x633	s	small s	0x73
ش	0x634	X	dollar sign	0x58
ص	0x635	S	capital s	0x53
ض	0x636	D	capital d	0x44
ط	0x637	T	capital t	0x54
ظ	0x638	Z	capital z	0x5a
ع	0x639	ç	small c-cedilla	0xe7
غ	0x63a	g	small g	0x67
ف	0x641	f	small f	0x46
ق	0x642	q	small q	0x71
ك	0x643	k	small k	0x6b

ل	0x644	l	small l	0x6c
م	0x645	m	small m	0x6d
ن	0x646	n	small n	0x6e
ه	0x647	h	small h	0x68
و	0x648	w	small w	0x77
ى	0x649	Y	capital y	0x59
ي	0x64a	y	small y	0x79
◌َ	0x64e	a	small a	0x61
◌ُ	0x64f	u	small u	0x75
◌ِ	0x650	i	small i	0x69
◌ً	0x64b	â	small a- circumflex	0xe2
◌ٌ	0x64c	û	small u- circumflex	0xfb
◌ٍ	0x64d	î	small i-circumflex	0xee
◌ّ	0x651	~	tilde	0x7e
◌ْ	0x652	·	interpunct	0xb7
-	0x640	¯	macron	0xaf

TABLE 30 Codification of Arabic characters and transliteration characters

Appendix B: Quantitative data extracted from Jabalín lexicons

In this appendix we show and briefly comment on some quantitative data extracted from the Jabalín lexicons—the lexicon of lemmas and the lexicon of inflected forms—to shed some light onto some of the linguistic properties of Arabic verbal system, and to examine some of the hypotheses on Arabic morphology that some authors have proposed in literature. We do not intend to present a robust analysis of the data, as this would be beyond the scope of this work and would require much additional effort. However, we believe that the material created by this research project should be conveniently used to perform linguistic analyses in a forthcoming task. Some of the data may be used to provide evidence for several statements made in the introduction and, at the same time, the material itself presents some interesting data supporting the convenience of our model for describing the verbal system. Yet, we insist this is just a rough outline of the statistical analysis that can be conducted on the data. It obviously requires further endeavours.

The lexicon of lemmas comprises 15,452 lemmas and a total of 3,706 roots. On average, triliteral roots generate 4.52 lemmas, and quadriliterals just 1.79.

	No. lemmas	No. roots	% lemmas/root
Triliteral	14,597	3,229	4.52
Quadriliteral	855	477	1.79
TOTAL	15,452	3,706	-

TABLE 31 Number of lemmas and roots in the lexicon and percentage of lemmas per root

Each of the 15,452 verb lemmas from the lexicon of lemmas generate 109 forms, resulting in a total of 1,684,268, which form the lexicon of inflected forms.

Total no. lemmas	No. forms/lemma	Total forms
15,452	109	1,684,268

TABLE 32 Number of entries in the lexicon of inflected forms

In the following table we can see the productivity of each pattern according to triliteral and quadriliteral roots. In the column 'Occurrences', we have the absolute frequency of verbs corresponding to the indicated pattern. In the column '%roots', we can see the percentage of roots having such pattern.

Root	Pattern	Verb occurrences	% Roots
Triliteral	Iau	1386	42.9
	Iai	1058	32.8
	Iaa	579	17.9
	Iuu	283	8.8
	Iia	936	29.0
	Iii	16	0.5
	II	1811	56.1
	III	996	30.8
	IV	2060	63.8
	V	1745	54.0
	VI	856	26.5
	VII	513	15.9
	VIII	1345	41.6
	IX	67	2.1
	X	831	25.7
	XI	59	1.8
	XII	47	1.5
	XIII	7	0.2
	XIV	-	-
	XV	3	0.1
Quadriliteral	QI	399	83.6
	QII	385	80.7
	QIII	5	1.0
	QIV	66	13.8

TABLE 33 Frequency of patterns

The ten most productive patterns of triliteral roots in the system are, by order of importance: IV, II, V, VIII, Iau, Iai, III, Iia, VI and X. The most remarkable cases are pattern IV, present in 63.8% of triliteral roots; pattern II, present in 56.1% of triliteral

roots; and pattern V, present in 54% of triliteral roots. A robust semantic analysis would be necessary to know the reason for these preferences.

Patterns XI, XII, XIII, XIV and X are extremely rare and, accordingly, their frequency in the lexicon is very low. A more interesting case is pattern Iii, comprising just 0.4% of the pattern I verbs. All simple patterns are expected a priori to have a good number of examples, but this is not the case of pattern Iii; the percentage of verbs belonging to this category is largely incidental. The 16 verbs included in the category are: نعم, حسب, وثق, وحد, ورث, ورع, ورك, ورم, وري, وعق, ولي, ومق, وهب, وهن, يئس, يبس. We looked up all the verbs in Cortés' (1996) and Hans Wehrs' (1993) dictionaries. The verbs نعم, وحد, وري, وهب, يبس did not follow pattern Iii conjugation in any of the dictionaries and there were not verbal entries of the verbs ورك and وعق. From the rest, the majority show other types of conjugations apart from pattern Iii. Only the verbs ورث, ورم, ولي and ومق correspond exclusively to the pattern Iii conjugation[90]. Apart from يئس and يبس, the rest of the verbs share the characteristic of having a *waw* as the first radical. Versteegh (2007) noted this and also the fact that there is a considerable number of verbs which admit *kasra*—vowel i—as well as *damma*—vowel u—in the imperfective, such as قَفَلَ – يَقْفِلُ / يَقْفُلُ. This may indicate that the pattern is phonetically motivated.

In the case of quadriliteral verbs, the majority of quadriliteral roots have at least one of the patterns QI and QII.

In the table below, the productivity of patterns found in the Jabalín lexicon of lemmas is compared with data collected by other authors. The data has been taken from Danks (2011). Danks indicates that the data from McCarthy and Prince is taken from their article "Prosodic morphology and templatic morphology" (1990). In this article the authors claim that they collected the data from the Hans Wehr dictionary. Danks

[90] Wright also provides a list of verbs following the pattern Iii. He includes 7 of the 16 verbs listed above—وثق, ورث, ورع, ورم, وري, ولي and ومق—and a few more which belong to other conjugational classes but may follow the pattern Iii (Wright, p.78).

himself makes use of the Hans Wehr lexicon, but a more recent edition than the one used by McCarthy and Prince. Danks indicates that Al-Qahtani himself takes his data from "A dictionary of Arabic verbs" by himself (2003).

Pattern	Al-Qahtani	McCarthy & Prince	Danks	Jabalín lexicon
I	2,512	2,569	2,523	4,258
II	1,479	1,398	1,416	1,811
III	455	463	465	996
IV	926	951	938	2,060
V	940	1,025	953	1,745
VI	375	394	389	856
VII	251	260	267	513
VIII	590	621	606	1,345
IX	18	18	19	67
X	393	389	395	831
XI	2	2	2	59
XII	7	7	7	47
XIII	-	-	-	7
XIV	1	2	1	-
XV	-	-	-	3
TOTAL			7,981	
QI	296	-	294	399
QII	111	-	131	385
QIII	1	-	1	5
QIV	8	-	8	66
TOTAL			434	

TABLE 34 Comparative table of pattern frequencies al-Qahtani, McCarthy and Prince, Danks and Jabalín

Our data differ the most compared to the other authors mentioned, probably because our sources were much more different—McCarthy and Prince, and Danks, took the Hans Wehr dictionary as a lexicographic source. It is reasonable to believe that al-Qahtani did the same.

A more interesting issue is the relationships established amongst patterns. Traditionally, it has been assumed that simple verbs—i.e. patterns Iau, Iai, Iaa, Iuu, Iia and Iii—are the base form verbs from which derived verbs are generated. This means that derived verbs with no pattern I verb counterpart cannot logically derive from a simple verb. If the number of derived verbs with no pattern I is low, we can assume that they constitute an exception to the previous assumption. Yet, they comprise an interesting case of study.

We extracted the number of occurrences of patterns of triliteral roots lacking pattern I and quadriliteral roots lacking pattern QI. The number of results are interesting since they may show the grade of dependency one pattern has with pattern I. Below we have a list of the actual occurrences of verbs from roots with no pattern I or pattern QI.

Roots	Pattern	Verb occurrences
Triliteral without Iau, Iai, Iaa, Iuu, Iia, Iii	II	86
	III	38
	IV	106
	V	72
	VI	27
	VII	16
	VIII	48
	IX	3
	X	26
	XI	3
	XII	1
	XIII	1
Quadriliteral without QI	QII	49
	QIII	2
	QIV	33

TABLE 35 Frequency of verbs from roots without any pattern I and QI

The most striking data are patterns IV, II and V, which exhibit the highest number of occurrences in roots without any Pattern I form.; there are 106 verbs of pattern IV, 86 verbs of pattern II and 72 verbs of pattern V belonging to roots with no pattern I verb. Interestingly, these three patterns were found to be the most productive in the lexicon, as we have seen previously.

In the case of pattern II, we pointed out in the introduction that some verbs belonging to this pattern are reported to be the result of a reanalysis process from geminated imperfective forms. We referred to the work of Corriente (2004) on intensive pattern

forms from Andalusian Arabic which were reanalyzed as pattern II verbs (Corriente, 2004). The figures of pattern II occurrences non-dependent on a pattern I form may be related with this reanalysis process. Yet, obviously much larger research is needed, especially together with semantic analyses.

In the case of quadriliteral verbs, it is not expected to have a strong correlation between pattern QI and the derived patterns QII, QIII and QIV, since these paradigms include many loanwords from other languages, so it is quite normal to have isolated verbs. Consequently, in many cases the existence of derived verbs without a correlated QI simple form does not involve any semantic implications.

The following table shows the number of roots according to its verbal lemmas productivity, i.e. the number of roots with only one verb, roots with two verbs, roots with three verbs, and so on.

Root	No. patterns/root	No. roots	% Roots
Triliteral	1	429	13.3
	2	461	14.3
	3	177	5.5
	4	487	15.1
	5	558	17.3
	6	443	13.7
	7	310	9.6
	8	210	6.5
	9	101	3.1
	10	38	1.2
	11	11	0.3
	12	5	0.2
Quadriliteral	1	118	24.7
	2	342	71.7
	3	15	3.1
	4	2	0.4

TABLE 36 Frequency of roots according to number of patterns per root

Almost 80% of triliteral roots yield between 1 to 6 verb lemmas and, if we increase the number up to 9 verbs per root, more than 98% of the roots are included. This means that roots with more than 6 verbs are uncommon and those with more than 9 roots are incidental. In the case of quadriliteral roots, more than 96% of the roots yield only 1 or 2 verbs. Roots with 3 or 4 verbs are extremely rare.

In the next table, we include the frequency of geminated verbs from triliteral roots, i.e., verbs in which the second and third radicals of the root are identical. In the case

of geminated quadriliteral verbs, the first and second radicals of the root are the same as the third and the fourth. Thus, these verbs are formed by reduplication.

Roots	Pattern	Verb occurrence
Triliteral roots that meet second rad. = third rad.	Iau	189
	Iai	101
	Iaa	13
	Iuu	8
	Iia	66
	II	164
	III	80
	IV	171
	V	133
	VI	84
	VII	53
	VIII	137
	X	100
Quadriliteral roots that meet first+second rad. = third+fourth rad.	QI	158
	QII	149

TABLE 37 Frequency of verbs from geminated roots

In triliteral geminated roots, the most productive patterns are Iau, IV, II, VIII, V, all being amongst the highest productive patterns of the total of triliteral roots. In the case of quadriliteral roots, the number of reduplicated roots compared to the total number of quadriliteral roots is very high. There are 158 verbs of pattern QI whose root is made by reduplication and, 149 verbs of pattern QII. If we consider all quadriliteral roots, we saw in table 33, that there were 399 verbs of pattern QI and 149 verbs of pattern QII. This means that 40% of QI verbs and 39% of QII verbs are built on a reduplicated root.

In the introduction, section 3, we saw that guttural consonants can affect their phonological context. Various authors (Versteegh et al., 2007; Wright, 2007; McCarthy, 1994; Danks, 2011; Watson, 2007) have reported the fact that guttural consonants—recall that the guttural class was composed by uvulars خ *x* and غ *g*, pharyngeals ح *H* and ع *Ç* and lagyngeals ء ç and ه *h*—tend to lower adjacent vowels due to their pharyngeal characteristic. As a result, many *a* vowels are derived from assimilated *i* or *u*. In this sense, Brame (1970) and later McCarthy (1994) claimed—and McCarthy endorsed this idea with data from the Hans Wehr dictionary—that pattern Iaa does not represent a real conjugational class, but it corresponds to Iai or Iau patterns whose middle vowel has lowered to *a* by the effect of a guttural consonant in second or third radical position. In the table below, we compare the frequencies of the guttural consonants in the three radical positions of Pattern Iaa against the rest of simple patterns, i.e., Iau, Iai, Iuu, Iia, Iii.

consonant	Pattern Iaa						Patterns Iau, Iai, Iuu, Iia, Iii					
	first radical		second radical		third radical		first radical		second radical		third radical	
	abs	%	abs	%	abs	%	abs	%	abs	%	abs	%
ب b	33	5.7	26	4.5	25	4.3	95	3.9	149	6.2	146	6.0
ف f	41	7.1	22	3.8	19	3.3	110	4.5	115	4.7	130	5.4
م m	56	9.7	20	3.5	22	3.8	113	4.7	135	5.6	174	7.2
ل l	36	6.2	35	6.0	29	5.0	82	3.4	161	6.6	202	8.3
ر r	37	6.4	38	6.6	45	7.8	126	5.2	227	9.4	253	10.4
ن n	59	10.2	16	2.8	15	2.6	204	8.4	93	3.8	129	5.3
ت t	2	0.3	6	1.0	5	0.9	27	1.1	68	2.8	32	1.3
د d	29	5.0	27	4.7	15	2.6	63	2.6	91	3.8	130	5.4
س s	33	5.7	14	2.4	12	2.1	113	4.7	84	3.5	95	3.9
ز z	19	3.3	14	2.4	6	1.0	58	2.4	76	3.1	47	1.9
ط T	11	1.9	14	2.4	12	2.1	49	2.0	66	2.7	71	2.9
ض D	12	2.1	14	2.4	7	1.2	35	1.4	47	1.9	40	1.7
ص S	16	2.8	9	1.6	12	2.1	61	2.5	58	2.4	39	1.6
ظ Z	4	0.7	1	0.2	3	0.5	6	0.2	13	0.5	12	0.5
ث þ	2	0.3	1	0.2	5	0.9	28	1.2	31	1.3	33	1.4
ذ ð	10	1.7	4	0.7	2	0.3	24	1.0	40	1.7	9	0.4
ش X	34	5.9	13	2.2	8	1.4	98	4.0	75	3.1	58	2.4
ج j	32	5.5	13	2.2	8	1.4	84	3.5	92	3.8	58	2.4
ك k	20	3.5	9	1.6	9	1.6	94	3.9	71	2.9	44	1.8
ق q	29	5.0	17	2.9	13	2.2	134	5.5	92	3.8	121	5.0
خ x	12	2.1	19	3.3	22	3.8	105	4.3	40	1.7	26	1.1
غ g	1	0.2	23	4.0	11	1.9	83	3.4	31	1.3	16	0.7
ح H	6	1.0	58	10.0	76	13.1	132	5.5	54	2.2	62	2.6
ع ç	4	0.7	64	11.1	116	20.0	161	6.6	67	2.8	84	3.5
ه h	15	2.6	62	10.7	11	1.9	85	3.5	71	2.9	25	1.0
ء c	3	0.5	25	4.3	54	9.3	85	3.5	24	1.0	38	1.6
و w	20	3.5	8	1.4	5	0.9	158	6.5	216	8.9	167	6.9
ي y	3	0.5	7	1.2	12	2.1	9	0.4	135	5.6	181	7.5
Total	566						2422					

TABLE 38 Frequency of guttural consonants in the three radical positions from Pattern Iaa and the rest of Patterns I.

Appendixes

The relative frequencies of guttural consonants in each group have been shaded to stand out. There seem to be a larger frequency of gutturals in second and third positions of the root in pattern Iaa, compared to the rest of simple patterns. This gives more weight to the idea that the morphological class of pattern Iaa is phonologically motivated by a process of vowel harmony around gutturals.

Apart from extracting data from the lexicon of lemmas, we analyzed the syllabic structure of the word-forms in the Jabalín lexicon of inflected forms. The syllabic analysis was carried out using the traditional counting system for prosody described by al-Khalil. Additionally, each form is associated with a weight, according to the counting algorithm. The algorithm was based on table 5, where we classified the syllables according to their expected weights. Below, we describe the algorithm created to perform the conversion.

1. We normalize the madda and shadda graphic symbols by two rules:
   (a) Ã -> ÁaA
   (b) C~a -> C·Ca ('C' stands for any consonant)

2. We convert each consonant plus a sukun into 0 (*sakin* letters). Long vowel letters *A*, *w*, *y* and *Y* are as well converted to 0 (*mamdood* letters). At the same time, all consonants supporting a short vowel are turned into 1 (*mutaharrik* letters).

3. We apply the counting substitutions to the forms sequentially:
   (a) 10 => 2
   (b) 12 => 3
   (c) 22 => 4

4. The resulting weights are converted into their syllable type: L (light), H (heavy), or sets LH and HH. As a final step, the sequence 'H0', is substituted by SH for super-heavy syllable. Below, a comprehensive table showing all the equivalences is given.
   (a) 4 => HH
   (b) 3 => LH
   (c) 2 => H
   (d) 1 => L
   (e) H0 => SH

Weight symbols	Accumulative conversion	Total weight	Sequence of syllables
1	1	1	L
10	10	2	H
110	12	3	LH
1010	22	4	HH

TABLE 39 Accumulative counting conversions into syllables

In the following tables we have the total weight and syllabic structures of the perfective and imperfective verb lemmas from the Jabalín lexicon of inflected forms classified by pattern.

Perfective lemmas				Imperfective lemmas		
Pattern	Total weight	Syllabic structure	Frequency	Total weight	Syllabic structure	Frequency
Iau	3	LLL	839	4	HLL	839
	3	HL	381	4	HH	167
	3	LH	167	4	LHL	381
Iai	3	HL	234	3	LH	16
	3	LH	135	4	HLL	580
	3	LLL	690	3	LLL	110
				4	HH	119
				4	LHL	234
Iaa	3	LLL	536	4	LHL	27
	3	LH	18	4	HH	17
	3	HL	26	4	HLL	536
Iuu	3	HL	8	4	LHL	8
	3	LLL	276	4	HH	4
				4	HLL	272
Iia	3	LLL	861	4	LHL	76
	2	LL	2	4	HH	90
	3	HL	74	4	HLL	771
Iii	3	LLL	17	4	HLL	5
				3	LH	2
				3	LLL	10
II	4	HLL	1633	5	LHLL	1633
	4	HH	179	5	LHH	179
III	3	SHL	77	5	LHLL	783
	4	HH	137	5	LHH	137
	4	HLL	783	4	LSHL	77
IV	4	HH	245	4	HH	245
	4	LHL	379	4	LHL	379

	4	HLL	1436	4	HLL	1436
V	5	LHLL	1576	6	LLHLL	1576
	5	LHH	170	6	LLHH	170
VI	5	LHLL	665	7	LHHLL	1
	5	LHH	111	5	LLSHL	83
	6	HHLL	1	6	LLHH	109
	4	LSHL	81	6	LLHLL	665
VII	5	HHL	113	5	HLH	36
	5	HLLL	366	5	HHL	113
	5	HLH	36	5	HLLL	366
VIII	5	HLLL	937	5	HLLL	940
	5	HHL	256	5	HHL	253
	4	HLL	2	5	LHLL	2
	5	HLH	151	5	HLH	151
IX	5	HHL	63	5	HLLL	1
	5	HLLL	1	5	HHL	63
	5	HLH	4	4	HH	4
X	6	HLHL	218	6	LHHL	2
	5	HHL	2	6	HHLL	505
	6	HHLL	505	6	HLHL	218
	6	HHH	107	6	HHH	107
XI	5	HSHL	58	5	HSHL	58
	6	HHLL	1	6	HHLL	1
	6	HHH	1	6	HHH	1
XII	6	HHH	16	6	HHH	16
	6	HHLL	32	6	HHLL	32
XIII	4	HH	1	5	LHH	1
	6	HLHL	7	6	HLHL	7
XV	6	HHLL	4	6	HHH	4
QI	4	HLL	399	5	LHLL	399
	4	HH	1	5	LHH	1

QII	5	LHLL	373	6	LLHLL	373
	5	LHH	11	6	LLHH	11
QIII	6	HHH	2	6	HHH	2
	6	HHLL	4	6	HHLL	4
QIV	6	HLHL	67	6	HLHL	67

TABLE 40 Syllabic structure and total weight of verbal patterns
H: heavy, L: light; SH; superheavy

As a general rule, we can see that each pattern tends to adjust its syllabic structure to a specific measure both in perfective and imperfective constructions. For example, all perfective lemmas from pattern Iau adjust its weight to 3, but through different syllabic structures: the majority of the verbs choose the structure LLL, but not a few follow the constructions HL and LH. The interesting idea here is that, without regarding the combination of syllables, each pattern tries to adjust itself to the same syllabic weight. Of course, there are several exceptional cases in which one form shows a different weight, either higher or lower, but these cases have a low frequency of occurrence, thus leaving them as normal exceptions. For instance, in pattern VIII perfective, there is a vast majority of verbs adjusting to the structures HLLL, HHL and HLL, all getting a weight score of 5. There are just two cases following a structure HLL, whose score is 4.

The idea behind this data is that Arabic verbs seem to adhere to an organized system of syllabic structures. Thus, it seems that a formalization of Arabic verbs based on syllabic principles is an adequate procedure to describe the Arabic verbal system.

Appendix C: Jabalín tagset and ElixirFM equivalence

POSITION	1	2	3	4	5	6	7
FEATURES	**POS**	**Aspect/tense**	**Voice**	**Mood**	**Person**	**Number**	**Gender**
VALUES	V	P	A	N	1	S	M
		I	P	S	2	D	F
				Y	3	P	N
				M			

Aspect/Tense	P	perfective
	I	imperfective
Voice	A	active
	P	passive
Mood	N	indicative
	S	subjunctive
	Y	jussive
	M	imperative
Person	1	first person
	2	second person
	3	third person
Gender	M	masculine
	F	feminine
	N	*not marking gender*
Number	S	singular
	D	dual
	P	plural

ElixirFM	Jabalín	inflectional information
VP-A-1MS--	VPAN1SN	perfective active indicative first person singular
VP-A-1MP--	VPAN1PN	perfective active indicative first person plural
VP-A-2MS--	VPAN2SM	perfective active indicative second person singular masculine
VP-A-2FS--	VPAN2SF	perfective active indicative second person singular feminine
VP-A-2MD--	VPAN2DN	perfective active indicative second person dual
VP-A-2MP--	VPAN2PM	perfective active indicative second person plural masculine
VP-A-2FP--	VPAN2PF	perfective active indicative second person plural feminine
VP-A-3MS--	VPAN3SM	perfective active indicative third person singular masculine
VP-A-3FS--	VPAN3SF	perfective active indicative third person singular feminine
VP-A-3MD--	VPAN3DM	perfective active indicative third person dual masculine
VP-A-3FD--	VPAN3DF	perfective active indicative third person dual feminine
VP-A-3MP--	VPAN3PM	perfective active indicative third person plural masculine
VP-A-3FP--	VPAN3PF	perfective active indicative third person plural feminine
VP-P-1MS--	VPPN1SN	perfective passive indicative first person singular
VP-P-1MP--	VPPN1PN	perfective passive indicative first person plural
VP-P-2MS--	VPPN2SM	perfective passive indicative second person singular masculine
VP-P-2FS--	VPPN2SF	perfective passive indicative second person singular feminine
VP-P-2MD--	VPPN2DN	perfective passive indicative second person dual
VP-P-2MP--	VPPN2PM	perfective passive indicative second person plural masculine
VP-P-2FP--	VPPN2PF	perfective passive indicative second person plural feminine
VP-P-3MS--	VPPN3SM	perfective passive indicative third person singular masculine
VP-P-3FS--	VPPN3SF	perfective passive indicative third person singular feminine
VP-P-3MD--	VPPN3DM	perfective passive indicative third person dual masculine
VP-P-3FD--	VPPN3DF	perfective passive indicative third person dual feminine
VP-P-3MP--	VPPN3PM	perfective passive indicative third person plural masculine
VP-P-3FP--	VPPN3PF	perfective passive indicative third person plural feminine
VIIA-1MS--	VIAN1SN	imperfective active indicative first person singular
VIIA-1MP--	VIAN1PN	imperfective active indicative first person plural
VIIA-2MS--	VIAN2SM	imperfective active indicative second person singular masculine
VIIA-2FS--	VIAN2SF	imperfective active indicative second person singular feminine
VIIA-2MD--	VIAN2DN	imperfective active indicative second person dual
VIIA-2MP--	VIAN2PM	imperfective active indicative second person plural masculine
VIIA-2FP--	VIAN2PF	imperfective active indicative second person plural feminine

ElixirFM	Jabalín	inflectional information
VIIA-3MS--	VIAN3SM	imperfective active indicative third person singular masculine
VIIA-3FS--	VIAN3SF	imperfective active indicative third person singular feminine
VIIA-3MD--	VIAN3DM	imperfective active indicative third person dual masculine
VIIA-3FD--	VIAN3DF	imperfective active indicative third person dual feminine
VIIA-3MP--	VIAN3PM	imperfective active indicative third person plural masculine
VIIA-3FP--	VIAN3PF	imperfective active indicative third person plural feminine
VIIP-1MS--	VIPN1SN	imperfective passive indicative first person singular
VIIP-1MP--	VIPN1PN	imperfective passive indicative first person plural
VIIP-2MS--	VIPN2SM	imperfective passive indicative second person singular masculine
VIIP-2FS--	VIPN2SF	imperfective passive indicative second person singular feminine
VIIP-2MD--	VIPN2DN	imperfective passive indicative second person dual
VIIP-2MP--	VIPN2PM	imperfective passive indicative second person plural masculine
VIIP-2FP--	VIPN2PF	imperfective passive indicative second person plural feminine
VIIP-3MS--	VIPN3SM	imperfective passive indicative third person singular masculine
VIIP-3FS--	VIPN3SF	imperfective passive indicative third person singular feminine
VIIP-3MD--	VIPN3DM	imperfective passive indicative third person dual masculine
VIIP-3FD--	VIPN3DF	imperfective passive indicative third person dual feminine
VIIP-3MP--	VIPN3PM	imperfective passive indicative third person plural masculine
VIIP-3FP--	VIPN3PF	imperfective passive indicative third person plural feminine
VISA-1MS--	VIAS1SN	imperfective active subjunctive first person singular
VISA-1MP--	VIAS1PN	imperfective active subjunctive first person plural
VISA-2MS--	VIAS2SM	imperfective active subjunctive second person singular masculine
VISA-2FS--	VIAS2SF	imperfective active subjunctive second person singular feminine
VISA-2MD--	VIAS2DN	imperfective active subjunctive second person dual
VISA-2MP--	VIAS2PM	imperfective active subjunctive second person plural masculine
VISA-2FP--	VIAS2PF	imperfective active subjunctive second person plural feminine
VISA-3MS--	VIAS3SM	imperfective active subjunctive third person singular masculine
VISA-3FS--	VIAS3SF	imperfective active subjunctive third person singular feminine
VISA-3MD--	VIAS3DM	imperfective active subjunctive third person dual masculine
VISA-3FD--	VIAS3DF	imperfective active subjunctive third person dual feminine
VISA-3MP--	VIAS3PM	imperfective active subjunctive third person plural masculine
VISA-3FP--	VIAS3PF	imperfective active subjunctive third person plural feminine
VISP-1MS--	VIPS1SN	imperfective passive subjunctive first person singular

ElixirFM	Jabalín	inflectional information
VISP-1MP--	VIPS1PN	imperfective passive subjunctive first person plural
VISP-2MS--	VIPS2SM	imperfective passive subjunctive second person singular masculine
VISP-2FS--	VIPS2SF	imperfective passive subjunctive second person singular feminine
VISP-2MD--	VIPS2DN	imperfective passive subjunctive second person dual
VISP-2MP--	VIPS2PM	imperfective passive subjunctive second person plural masculine
VISP-2FP--	VIPS2PF	imperfective passive subjunctive second person plural feminine
VISP-3MS--	VIPS3SM	imperfective passive subjunctive third person singular masculine
VISP-3FS--	VIPS3SF	imperfective passive subjunctive third person singular feminine
VISP-3MD--	VIPS3DM	imperfective passive subjunctive third person dual masculine
VISP-3FD--	VIPS3DF	imperfective passive subjunctive third person dual feminine
VISP-3MP--	VIPS3PM	imperfective passive subjunctive third person plural masculine
VISP-3FP--	VIPS3PF	imperfective passive subjunctive third person plural feminine
VIJA-1MS--	VIAY1SN	imperfective active jussive first person singular
VIJA-1MP--	VIAY1PN	imperfective active jussive first person plural
VIJA-2MS--	VIAY2SM	imperfective active jussive second person singular masculine
VIJA-2FS--	VIAY2SF	imperfective active jussive second person singular feminine
VIJA-2MD--	VIAY2DN	imperfective active jussive second person dual
VIJA-2MP--	VIAY2PM	imperfective active jussive second person plural masculine
VIJA-2FP--	VIAY2PF	imperfective active jussive second person plural feminine
VIJA-3MS--	VIAY3SM	imperfective active jussive third person singular masculine
VIJA-3FS--	VIAY3SF	imperfective active jussive third person singular feminine
VIJA-3MD--	VIAY3DM	imperfective active jussive third person dual masculine
VIJA-3FD--	VIAY3DF	imperfective active jussive third person dual feminine
VIJA-3MP--	VIAY3PM	imperfective active jussive third person plural masculine
VIJA-3FP--	VIAY3PF	imperfective active jussive third person plural feminine
VIJP-1MS--	VIPY1SN	imperfective passive jussive first person singular
VIJP-1MP--	VIPY1PN	imperfective passive jussive first person plural
VIJP-2MS--	VIPY2SM	imperfective passive jussive second person singular masculine
VIJP-2FS--	VIPY2SF	imperfective passive jussive second person singular feminine
VIJP-2MD--	VIPY2DN	imperfective passive jussive second person dual
VIJP-2MP--	VIPY2PM	imperfective passive jussive second person plural masculine
VIJP-2FP--	VIPY2PF	imperfective passive jussive second person plural feminine
VIJP-3MS--	VIPY3SM	imperfective passive jussive third person singular masculine

ElixirFM	Jabalín	inflectional information
VIJP-3FS--	VIPY3SF	imperfective passive jussive third person singular feminine
VIJP-3MD--	VIPY3DM	imperfective passive jussive third person dual masculine
VIJP-3FD--	VIPY3DF	imperfective passive jussive third person dual feminine
VIJP-3MP--	VIPY3PM	imperfective passive jussive third person plural masculine
VIJP-3FP--	VIPY3PF	imperfective passive jussive third person plural feminine
VCJ---MS--	VIAM2SM	imperative active second person singular masculine
VCJ---FS--	VIAM2SF	imperative active second person singular feminine
VCJ---MD--	VIAM2DN	imperative active second person dual
VCJ---MP--	VIAM2PM	imperative active second person plural masculine
VCJ---FP--	VIAM2PF	imperative active second person plural feminine

Appendix D: Jabalín pattern-code equivalences

Arabic Pattern	Convention for pattern	Jabalín code
فَعَلَ _ُ	Iau	00L0001
فَعَلَ _ِ	Iai	00L0000
فَعَلَ _َ	I aa	00L0002
فَعُلَ _ُ	Iuu	00L0101
فَعِلَ _َ	Iia	00L0202
فَعِلَ _ِ	Iii	00L0200
فعّل	II	20H0010
فاعل	III	30H0010
أفعل	IV	03H0010
تفعّل	V	20H1002
تفاعل	VI	30H1002
انفعل	VII	01L0000
افتعل	VIII	02L0000
افعلّ	IX	10L0000
استفعل	X	04H0000
افعالّ	XI	15H0000
افعوعل	XII	56H0000
افعوّل	XIII	07H0000
افعنلل	XIV	18H0000
افعنلى	XV	48H0000
فعلل	QI	00H0010
تفعلل	QII	00H1002
افعنلل	QIII	08H0000
افعللّ	QIV	10H0000

Appendix E: Python code of Jabalín

jabalin.py

```
import sys
import os
import glob
import shutil
import datetime
from GenerationVerbs import *
from BdD.BdDalicia import *
from BdD.BdDElixir import *
from eval_elixir import *
from preparacionEval import *
from Elixir_Inflect_Derivation import *
from actualizarBdD import *
from DataExtraction import *


import servidor_apli

def copiaSeguridad(file_srt, dst):
    principal, fichero=file_srt.split('\\')
    fich, ext= fichero.split('.')
    all_files = glob.glob(principal+"\*."+ext)
    if file_srt in all_files:
        shutil.copy(file_srt, dst)
        now = datetime.datetime.today()
        date_time= now.strftime("%Y_%m_%d_T%H-%M-%S")
        os.rename(dst+'\\'+fichero,
                  dst+'\\'+fich+'_'+date_time+'.'+ext)


def imprime_info():
    print("Opciones del Analizador Jabalín:")
    print("\n_____________________________________________________________________")
    print('código'.rjust(12))
    print('\tA\tAYUDA')
    print("\n_____________________________________________________________________")
    print("EJECUCIÓN AUTOMÁTICA\n")
    print('código'.rjust(12))
    print('\t1\tGENERAR LEXICÓN DE VERBOS')
    print('\t2\tGENERAR BASE DE DATOS DEL LEXICÓN JABALÍN')
```

```
    print('\t3\tGENERAR BASE DE DATOS DEL LEXICÓN ELIXIR')
    print('\t4\tEVALUAR LEXICÓN')
    print('\t5\tGENERAR BASE DE DATOS DESDE EL PRINCIPIO (1, 2 y 4)')
    print('\t6\tPASAR TEST A LA GENERACIÓN DEL LEXICÓN')
    print('\t7\tEVALUAR FRECUENCIAS')

    print("\n_____________________________________________________________")
    print("EJECUCIÓN SEMIAUTOMÁTICA: controlando el nombre de los
ficheros\n")
    print('código'.rjust(12))
    print('\t8\tGENERAR LEXICÓN DE VERBOS')

    print("\n_____________________________________________________________")
    print("MISCELÁNEA:\n")
    print('código'.rjust(12))
    print('\t9\tEXTRAER LA DERIVACIÓN NOMINAL DEL ELIXIR')
    print('\t10\tEXTRAER LEXICÓN DEL ELIXIR_FM')
    print('\t11\tPREPARACION DEL LEXICÓN DE VERBOS DEL ELIXIR_FM PARA
EVALUACIÓN')
    print('\t12\tMODIFICAR BASE DE DATOS JABALÍN')
    print('\t13\tGENERAR AUTOMÁTICAMENTE LA BdD JABALÍN Y ACTUALIZARLA')
    print('\t14\tEXTRACT QUANTITATIVE DATA FROM LEXICONS')
    print('\t0\tSALIR')
    print("\n_____________________________________________________________")


    print('\n\nMARQUE EL CÓDIGO DE LA ACCIÓN QUE DESEE REALIZAR (1 ó 2 ó
3...):\n> ', end="")

imprime_info()
for line in iter(sys.stdin.readline, ""):
    codigo=line.strip()
    actualizar='NO'


    if codigo== 'A':
        import webbrowser
        l=os.getenv('PROGRAMFILES')

        ffcommand = "c:/"+l[3:]+"/mozilla firefox/firefox.exe %s &"
        webbrowser.get(ffcommand).open_new("ayuda.htm")
        print('Información abierta en su navegador firefox.')
        imprime_info()

    if codigo=='0':
        print('ADIOS!!  :@ Te echaré de menos, sniff :@ oink!!')
```

```
        break

    if codigo=='13':
        print("Preparados para actualizar la Base de Datos Jabalín...")
        print("Haciendo copia de seguridad de la base de datos en
ResultadosBdD\CopiaSeguridadBdD...")

copiaSeguridad('ResultadosBdD\lexiconVerbsdata_Jabalin.sqlite','ResultadosBd
D\CopiaSeguridadBdD')

        print("Renombrando la base de datos antigua...")
        os.rename('ResultadosBdD\lexiconVerbsdata_Jabalin.sqlite',

'ResultadosBdD\lexiconVerbsdata_Jabalin_antigua.sqlite')
        print("Generando automáticamente la base de datos Jabalín...")
        actualizar='SI'
        codigo='5'

    if codigo=='5':
        print("Haciendo todo el proceso de evaluación, incluida la
generación de verbos")

    if codigo=='1' or codigo=='5':
        print("Haciendo copia de seguridad de la base de datos en
ResultadosFile\CopiaSeguridadFiles...")

copiaSeguridad('ResultadosFile\lexiconVerbs_jabalin.txt','ResultadosFile\Cop
iaSeguridadFiles')
        print("Generando 'lexiconVerbs_jabalin.txt' en la carpeta
ResultadosFile...")
        GV.generation_verbs('GenerationVerbs\lexicon_lemas_jabalin.txt',
'ResultadosFile\lexiconVerbs_jabalin.txt')
        print('terminado')
        if codigo=='1':
            imprime_info()

    if codigo=='2' or codigo=='5':
        print("Generando Base de Datos Jabalín")
        print("Haciendo copia de seguridad de la base de datos en
ResultadosBdD\CopiaSeguridadBdD...")

copiaSeguridad('ResultadosBdD\lexiconVerbsdata_Jabalin.sqlite','ResultadosBd
D\CopiaSeguridadBdD')
        print('borrando ya la existente')
        all_files = glob.glob("ResultadosBdD\*.sqlite")
        if 'ResultadosBdD\lexiconVerbsdata_Jabalin.sqlite' in all_files:
```

```
            os.remove("ResultadosBdD\lexiconVerbsdata_Jabalin.sqlite")
        print('creando la nueva')
        createDBtablesAlicia.crear_jabalin()
        print('inicializándola')
        initDBalicia.inicializar_jabalin()
        if codigo=='2':
            print('terminado')
            imprime_info()

    if codigo=='3':
        print("Generando Base de Datos Elixir")
        print("Haciendo copia de seguridad de la base de datos en
ResultadosBdD\CopiaSeguridadBdD...")

copiaSeguridad('ResultadosBdD\lexiconVerbsdata_Elixir.sqlite','ResultadosBdD
\CopiaSeguridadBdD')
        print('borrando la existente')
        all_files = glob.glob("ResultadosBdD\*.sqlite")
        if 'ResultadosBdD\lexiconVerbsdata_Elixir.sqlite' in all_files:
            os.remove("ResultadosBdD\lexiconVerbsdata_Elixir.sqlite")
        print('creando la nueva')
        createDBtablesElixir.crear_elixir()
        print('inicializándola')
        initDBelixir.inicializar_elixir()
        print('terminado')
        imprime_info()

    if codigo=='4' or (codigo=='5' and actualizar!='SI'):
        print("Evaluando lexicón de verbos Jabalín")
        eval.evaluacion()
        print('terminado')
        imprime_info()

    if actualizar=='SI':
        actualizar='NO'
        print("actualizando Base de Datos...")
        actualizarBdD.actualizarBdD()
        print("borrando base de datos antigua...")
        if 'ResultadosBdD\lexiconVerbsdata_Jabalin_antigua.sqlite' in
all_files:

os.remove("ResultadosBdD\lexiconVerbsdata_Jabalin_antigua.sqlite")
        print('terminado')
        imprime_info()

    if codigo=='6':
```

```
        print("el 'antiguo file_verbs_test.txt' se debe llamar
'jabalin_verbs_test.txt'")
        print("Pasando el test al lexicón...")
        GV.generation_verbs('GenerationVerbs\lexicon_lemas_test.txt',
'ResultadosFile\lexiconVerbs_jabalin_test.txt')
        print("evaluando")
        verbs_tester.test_verbs()
        print("terminado el test")
        imprime_info()

    if codigo=='7':
        print("Evaluando frecuencias...")
        print("Generando 'lexiconVerbs_freq.txt'...")
        limpia_lema_freq.limpiar_lema_freq()
        GV.generation_verbs('ResultadosFile\lexicon_lemas_freq.txt',
'ResultadosFile\lexiconVerbs_freq.txt')
        print("Analizando la Base de Datos jabalín...")
        eval.Vbs_mas_freq()
        print("terminado")
        imprime_info()

    if codigo=='8':
        nombre_f_in=input("Escriba el nombre del fichero de entrada: ")
        nombre_f_out=input("Escriba el nombre del fichero de salida: ")
        print("Generando "+ nombre_f_out +"en la carpeta ResultadosFile...")
        GV.generation_verbs("GenerationVerbs'\'"+ nombre_f_in,
"ResultadosFile'\'"+nombre_f_out)
        print('terminado')
        imprime_info()

    if codigo=='9':
        print("Derivación nominal del Elixir...")
        print("Haciendo copia de seguridad del fichero en
ResultadosFile\CopiaSeguridadFiles...")

copiaSeguridad('ResultadosFile\elixir_nominal_derivation.txt','ResultadosFil
e\CopiaSeguridadFiles')
        print("generando 'derive_elixir.txt' y usando el encode para generar
'derive_elixir_encod.txt' y procesarlo en 'elixir_nominal_derivation.txt'")
        succionador_Elixir.encode_derive()
        print('Los ficheros resultantes intermedios están en el path:
.\Elixir_Inflect_Derivation\ResultadosIntermediosElixir_FM ')
        print("El fichero 'elixir_nominal_derivation.txt' resultante está en
el path: .\ResultadosFile")
        print('terminado')
        imprime_info()
```

```
    if codigo=='10':
        print("Flexión del Elixir_FM...")
        print("Haciendo copia de seguridad del fichero en
ResultadosFile\CopiaSeguridadFiles...")

copiaSeguridad('ResultadosFile\lexiconVerbs_elixir_otakar.txt','ResultadosFi
le\CopiaSeguridadFiles')

        print("  1.- extrayendo la flexión del Elixir sin traducir en
'inflec_elixir.txt'")
        succionador_Elixir.inflected()

        print("  2.- traduciendo al Árabe con encode del Elixir_FM
'inflec_elixir.txtt' en 'inflec_elixir_encoded.txt'")
        succionador_Elixir.encode()

        print("  3.- generando el formato del fichero deseado, añadiendo el
código del lema en función de la coordenada,
'lexiconVerbs_elixir_otakar.txt'")
        succionador_Elixir.generate_file_Inflected_Verbs()
        succionador_Elixir.procesar_unknown()
        succionador_Elixir.incluye_lemas_code_unknown()

        print('Los ficheros resultantes intermedios están en el path:
.\Elixir_Inflect_Derivation\ResultadosIntermediosElixir_FM ')
        print('El fichero resultante está en el path: .\ResultadosFile')
        print('terminado')
        imprime_info()


    if codigo=='11':
        print("Preparando el lexicón de verbos del Elixir_FM
'lexiconVerbs_elixir_otakar.txt' para la evaluación...")
        print("  1.- sacando 'lexicon_lemas_eval.txt' común con jabalín")
        fixPatterns_MAIN.normaliza_lemaCode_elixir()
        print("  2.- traduciendo patter a código")
        fixPatterns_MAIN.extrae_lemas_eval()

        # borrando los ficheros intermedios
        all_files = glob.glob("*.txt")
        for f in all_files:
            os.remove(f)
        print('terminado')
        imprime_info()
```

```
    if codigo=='12':
        print("Preparados para modificiar la Base de Datos Jabalín...")
        print("Aparecerá una ventana del 'firefox' con la dirección
'localhost:8080'...")
        servidor_apli.arrancar()
        print('terminado')
        imprime_info()

    if codigo== '14':
        extractData_from_lexicons.quantitativeData_extractor()
```

GV.py

```
########################################################################
# MODULE             VERBAL GENERATION SYSTEM
########################################################################

import re
import sys

from GenerationVerbs import utilities
from GenerationVerbs import ID
from GenerationVerbs import PT
from GenerationVerbs import ED
from GenerationVerbs import vocalization
from GenerationVerbs import stem_adjustment
from GenerationVerbs import Inflec
from GenerationVerbs import phonotactics


def generation_verbs(filename, file_out):
    try:
        with open(filename, encoding='utf8') as file, open(file_out, 'w',
encoding='utf8') as generation_file:
            try:
                # FOR EACH LINE IT SAVES LEMMA, ROOT, CODE, INFLECTION AND
TAG
                for each_line in file:
                    if not utilities.delete_line(each_line):
                        (lema, root, code)=each_line.strip().split('\t',2)

                        # parseamos el código
                        dict_code=utilities.parse_code_verbs(code)
```

```
                        # 1.- INTERNAL DERIVATION
                        deriv_root = ID.Internal_derivation(root, \

dict_code['Internal derivation'])

                        # 2.- PROSODIC TEMPLATE
                        dict_RootInPT = PT.prosodic_template(deriv_root, \

dict_code['Template'] )

                        # 3.- EXTERNAL DERIVATION
                        deriv_RootInPT =
ED.External_derivation(dict_RootInPT, \

dict_code['External derivation'])

                        # 4.- VOCALIZATION
                        dict_act_pas_forms =
vocalization.generate_Active_and_Pasive(deriv_RootInPT, \

dict_code['Vocalization'])

                        # 5.- STEM ADJUSTMENTS
                        dict_syl_forms =
stem_adjustment.rules_stem_adjustment(dict_act_pas_forms, dict_code, root,
lema)

                        # 6.- INFLECTED SYSTEM
                        dict_inflec_forms =
Inflec.Inflectional_system(dict_syl_forms)


                        # 7.- PHONOTACTIC CONSTRAINTS AND ORTHOGRAPHIC
NORMALIZATION
                        final_forms =
phonotactics.phonotactic_rules(dict_inflec_forms, lema, root, dict_code)
                        utilities.printFile_forms_from_dictForms(lema, root,
code, final_forms, generation_file)

            except ValueError as verror1:
                print('Value error Principal: ' + str(verror1))

    except IOError as ioerr:
        print('File error: ' + str(ioerr))
        return(None)
```

ID.py

```
######################################################################
# MODULE             INTERNAL DERIVATION
######################################################################

import re
import sys

# This function applies the rules classified as lengthening operations of
Internal
# Derivation. The rules are applied to the root, which is passed as a
parameter
def ID_rules_lenghtening(root, cod_lenghtening):
    cod=int(cod_lenghtening)
    # the last character is duplicated
    if cod==1:
        root=re.sub(r"(.*)(.)", r"\1\2\2", root)

    # '̃' ('~') is added between the second and the third characters
    elif cod==2:
        root=re.sub(r"(..)(.*)", r"\1̃\2", root)

    # 'E' (lengthening mark) is added between the first and the second
characters.
    elif cod==3:
        root=re.sub(r"(.)(.*)", r"\1E\2", root)

    # 'E' is added at the end
    elif cod==4:
        root=re.sub(r"(.+)", r"\1E", root)

    # the second character is duplicated
    elif cod==5:
        root=re.sub(r"(.)(.)(.*)", r"\1\2\2\3", root)

    return (root)


# This function applies the rules classified as addition operations of
Internal
# Derivation. Again, the rules are applied to the root
```

```
def ID_rules_addition(root, cod_addition):
    cod=int(cod_addition)
    # 'ن' ('n') is added at the beginning
    if cod==1:
        root=re.sub(r"(.+)", r"1\ن", root)

    # 'ت' ('t') is added between the first and the second characters.
    elif cod==2:
        root=re.sub(r"(.)(.*)", r"\12\ت", root)

    # 'أ' ('Á') is added at the beginning
    elif cod==3:
        root=re.sub(r"(.+)", r"1\أ", root)

    # 'ست' ('st') is added at the beginning
    elif cod==4:
        root=re.sub(r"(.+)", r"1\ست", root)

    # 'E' is added between the second and the third characters.
    elif cod==5:
        root=re.sub(r"(..)(.*)", r"\1E\2", root)

    # 'و' ('w') is added between the second and the third characters.
    elif cod==6:
        root=re.sub(r"(..)(.*)", r"\12\و", root)

    # 'وو' ('ww') is added between the second and the third characters.
    elif cod==7:
        root=re.sub(r"(..)(.*)", r"\12\وو", root)

    # 'ن' ('n') is added between the second and the third characters.
    elif cod==8:
        root=re.sub(r"(..)(.*)", r"\12\ن", root)

    return (root)


# This function applies all the rules of the Internal Derivation
# to the root, which is passed as parameter
def Internal_derivation(root, cod_ID):

    root_id = ID_rules_lenghtening(root, cod_ID["lengthening"])
    root_id = ID_rules_addition(root_id, cod_ID["addition"])

    return(root_id)
```

PT.py

```
########################################################################
# MODULE             PROSODIC TEMPLATE
########################################################################

import re
import sys
from GenerationVerbs import utilities


# This function gets a codec specifing the Prosodic Template which has to be
applied, L or H,
# and returns a dictionary containing the templates for
Perfective,Imperfective and Imperative.
def select_prosodic_template(cod_template):
    dict_PT = {}
    if cod_template == 'L':
        dict_PT={'VP': 'FFVFWF', 'VI':'VFFFWF', 'VIAM':'FFFWF'}
    elif cod_template == 'H':
        dict_PT={'VP': 'FFVFFWF', 'VI':'VFFFFWF', 'VIAM':'FFFFWF'}
    return (dict_PT)


# This function gets as parameters the root+affixation and the prosodic
template
# It inserts the root+affixation in the template and returns the resulting
string, the stem

# NOTE Imperfective: VF5F4F3F2WF1 and sffn first VF5sffWn aster VssfWn
# NOTE Perfective and Imperative: F5F4VF3F2WF1 and sffn first F5sVffWn aster
sVsfWn
def apply_prosodic_template(der_root, template):
    # the order is inverted to apply the substitutions in the opposite
direction
    rev_temp=utilities.invertir(template)
    rev_root=utilities.invertir(der_root)

    # each character of the root+affixation is substituted by the Fs of the
template
    rev_temp_root=rev_temp
    for char in rev_root:
```

```
        rev_temp_root=rev_temp_root.replace('F', char, 1)

    # is there are remaiming Fs, they are removed from the stem
    rev_temp_root=rev_temp_root.replace('F', '')

    # again, we invert the form, so that the characters of the stem are in
the correct direction
    temp_root = utilities.invertir(rev_temp_root)

    return(temp_root)


# This function insterts the root+affixation characters in the templates,
and
# returns the dictionary with i-stem (VP), p-stem (VI), and m-stem (VIAM)
def apply_prosodic_all_templates(der_root, dict_PT):
    # p-stem
    dict_PT['VP'] = apply_prosodic_template(der_root, dict_PT['VP'])
    # i-stem
    dict_PT['VI'] = apply_prosodic_template(der_root, dict_PT['VI'])
    # m-stem
    dict_PT['VIAM'] = apply_prosodic_template(der_root, dict_PT['VIAM'])

    return(dict_PT)


# This function gets as parameters the root+affixation and the codec, so
that the prosodic template
# function can be applied. It returns a dictionary of templates with the
root+affixation replacements
def prosodic_template(der_root, cod_template):
    dict_PT = select_prosodic_template(cod_template)
    dict_PT_root = apply_prosodic_all_templates(der_root, dict_PT)

    return (dict_PT_root)
```

ED.py

```
#######################################################################
# MODULE             EXTERNAL DERIVATION
#######################################################################

import re
import sys
```

```
# This function gets all the pattern VIII stems, included in dict_T,
# and applies the corresponding rules if the cod_ED is '1'.
# It returns the dict_T dictionary modified.

def External_derivation(dict_T, cod_ED):
    cod = int(cod_ED)
    if cod == 1:
        # prefix -ت (-ta) is added to pattern VIII stems
        dict_T['VP']=re.sub(r"(.+)", r"1\ت", dict_T['VP'])
        dict_T['VIAM']=re.sub(r"(.+)", r"1\ت", dict_T['VIAM'])
        dict_T['VI']=re.sub(r"(.)(.+)", r"\12\ت", dict_T['VI'])

    return (dict_T)
```

vocalization.py

```
######################################################################
# MODULE             VOCALIZATION
######################################################################

import re
import sys

dict_vocalization = {'Perf W': {'0': 'َ', '1': 'ْ'2' ,''},\
                     'Imperf V': {'0': 'ُ', '1': ''}, \
                     'Imperf W': {'0': 'ِ', '1': 'ْ'2' ,''}}

# This function generates perfective active
def generate_VPA (perfective, cod_perfect):

    # vowel V default values
    VPA = perfective.replace('V', 'َ')
    # vowel W values in dictionary
    VPA = VPA.replace('W', dict_vocalization['Perf W'][cod_perfect])

    return(VPA)


# This function generates perfective passive
def generate_VPP (perfective):

    # vowel V default values
    VPP = perfective.replace('V', 'ُ')
```

```
    # vowel W default values
    VPP = VPP.replace('W', '़')

    return(VPP)


# This function generates imperfective active
def generate_VIA (imperfective, cod_imperfV, cod_imperfW):

    # vowel V values in dictionary
    VIA = imperfective.replace('V', dict_vocalization['Imperf
V'][cod_imperfV])
    # vowel W values in dictionary
    VIA = VIA.replace('W', dict_vocalization['Imperf W'][cod_imperfW])

    return(VIA)


# This function generates imperfective passive
def generate_VIP (imperfective):

    # vowel V default values
    VIP = imperfective.replace('V', 'ं')
    # vowel W default values
    VIP = VIP.replace('W', 'ं')

    return(VIP)


# This function generates imperative active
def generate_VIAMA (imperative, cod_imperfW):

    # vowel W values in dictionary
    VIAMA = imperative.replace('W', dict_vocalization['Imperf
W'][cod_imperfW])

    return(VIAMA)



# This function returns a dictionary with different stems. Stems vary in
tense/aspect and voice.
# Hence we have active p-stem, passive p-stem, active i-stem, passive i-stem
and active m-stem
# i.e. {'VI-P': 'udrhmam', 'VI-A': 'adrhmim', 'VP-A': 'drahmam', 'VP-P':
'druhmim', 'VIAM-A': 'drhmim'}
```

```
def generate_Active_and_Pasive (dict_tenses, cod_vocalization):

    dict_voc_tens={}
    dict_voc_tens['VP-A']=generate_VPA(dict_tenses['VP'],\
                                       cod_vocalization['Perf V2'])
    dict_voc_tens['VP-P']=generate_VPP(dict_tenses['VP'])
    dict_voc_tens['VI-A']=generate_VIA(dict_tenses['VI'],\
                                           cod_vocalization['Imperf V1'],
\
                                           cod_vocalization['Imperf V2'])
    dict_voc_tens['VI-P']=generate_VIP(dict_tenses['VI'])
    dict_voc_tens['VIAM']=generate_VIAMA(dict_tenses['VIAM'], \
                                                cod_vocalization['Imperf
V2'])
    return(dict_voc_tens)
```

stem_adjustment.py

```
######################################################################
# MODULE             PHONOTACTICS AT STEM LEVELS
######################################################################

import re
import sys

def list_rules_stem_adjustment(f, form, dict_cod, root, lema):

    # if ID addition is '3' the letter in second position is removed
    # from VIA and VIP forms
    forms = ['VI-A', 'VI-P']
    if (f in forms) & (dict_cod['Internal derivation']['addition']=='3'):
        form = re.sub(r"(.)(.)(.*)", r"\1\3", form)     # A1

    # assimilation of V (u) from the ta- prefix in perfective passive
template
    # patterms V, VI triliteral and II quadriliteral, ta -> tu
    # eg.: ta.fuw.çal -> tu.fuw.çil
    if (f=='VP-P')&(dict_cod['External derivation']=='1'):
        form=form.replace(form[1], 'ó', 1)              # A2


    # for tenmplate L, if a sequence CCC is found from the end to the
beginning,
```

```
    # it is substituted by CCaC
    if dict_cod['Template']== 'L':

form=re.sub(r"(.*)(?<=[Eئؤءيأهنمكلقفعغظطضصشسزرذدخحجثتبا])(.)(?<=[E
ئؤءيأهنمكلقفعغظطضصشسزرذدخحجثتبا])"
,\
                          r"\1\2ّ\3" , form)            # A3

    # for template H, if a sequence CCC is found from the end to the
beginning,
    # it is substituted by CaCC
    if dict_cod['Template']== 'H':

form=re.sub(r"(.*)(?<=[Eئؤءيأهنمكلقفعغظطضصشسزرذدخحجثتبا])(.)(?<=[E
ئؤءيأهنمكلقفعغظطضصشسزرذدخحجثتبا])"
,\
                    r"\1ّ\2\3" , form)                  # A4


    # ASSIMILATION OF INFIX -T- IN PATTERN VIII

    if dict_cod['Internal derivation']['addition']=='2':

        form = re.sub(r'^(.?)[ت[وي',r'\1تـ', form)      # B1 [wy]t -> t~ /
^.?_   eg تـتحد <- وتـحد
        form = re.sub(r'^(.?[ت([ثـذظط]', r'\1\1', form)  # B2 t -> ~ /
^[þdðTZ].?_   eg ظظنن <- ظتـنن
        form = re.sub(r'^(.?ت(ز', r'\1د', form)         # B3 t -> d / ^.?z_
eg زدوج <- زتـوج
        form = re.sub(r'^(.?[ت([صض]', r'\1ط', form)     # B4 t -> T /
^.?[DS]_   eg ضطرب <- ضترب

        if re.search(r'^ء', root):
            if re.search(r'^اتـ', lema):
                form = re.sub(r'^(.?)ءت',r'\1تـ', form) # B5  ct -> t~ /
^.?_   eg تـخذ <- ءتـخذ

    return(form)


def rules_stem_adjustment(dict_forms, dict_cod, root, lema):
    forms = ['VP-A', 'VP-P', 'VI-A', 'VI-P', 'VIAM']

    for f in forms:
        dict_forms[f] = list_rules_stem_adjustment(f, dict_forms[f],
dict_cod, root, lema)
```

```
    return (dict_forms)
```

Inflec.py

```
########################################################################
# MODULE             INFLECTIONAL SYSTEM
########################################################################

import re
import sys


# This function gets a list containing elements which are tuples, and
returns a dictionary
def pass_list_to_dictionary (list_tuples):

    dict={}
    for l in list_tuples:
        dict[l[1]]=l[0]
    return dict


# This function gets a form, a list of the inflectional suffixes, a table
with
# the partial tags of each suffix, and f, which indicates if the form is
perfective,
# imperfective or imperative. Returns a list of tuples: inflected form with
its tag
def STEM_plus_suf(form, list_cod, table_tag, f):
    list_inf_tag = [[form+table_tag[l], f+l] for l in list_cod]

    return (list_inf_tag)


# This function gets a form, a list of the prefixes, a table with the
partial tags
# of each suffix, and f, which indicates if the form is perfective,
imperfective
# or imperative. Returns a list of tuples: inflected form with its tag
def pre_plus_STEM(form, list_cod, table_tag, f):
    list_inf_tag = [[table_tag[l]+form, f+l] for l in list_cod]
    return (list_inf_tag)
```

```
# This function gets a form, a list of prefixes and suffixes, a table with
the partial
# tags of each suffixes, and f which, indicates if the form is perfective,
imperfective
# or imperative. Returns a list of tuples: inflected form with its tag
def pre_plus_STEM_plus_suf(form, list_cod, table_tag, f):
    list_inf_tag=[]
    for l in list_cod:
        (pre, suf)=table_tag[l].split('_')
        list_inf_tag.append([pre+form+suf, f+l])

    return (list_inf_tag)


# This function gets the dict with the forms and returns a dictionary with
the inflected forms
# for passive and active perfective as a list with pairs of inflected form
and its tag
def inflected_perfective_forms(dict_forms):
    table_perfective = {'N1SN': 'تُ',
                        'N1PN': 'نَE',
                        'N2SM': 'تَ',
                        'N2SF':'تِ',
                        'N2DN': 'تُمَE',
                        'N2PM':'تُم',
                        'N2PF': 'تُنَّ',
                        'N3SM': 'َ',
                        'N3SF': 'تَ',
                        'N3DM': 'َE',
                        'N3DF': 'تَ َE',
                        'N3PM': 'ُEا',
                        'N3PF': 'نَ'}
    cod = table_perfective.keys()

    list_act = STEM_plus_suf(dict_forms['VP-A'], cod,
table_perfective,'VPA')
    list_pas = STEM_plus_suf(dict_forms['VP-P'], cod,
table_perfective,'VPP')


    return({'Active': pass_list_to_dictionary(list_act),
            'Pasive': pass_list_to_dictionary(list_pas)})
```

```
# This function gets the form VIAM (imperatives) and returns a list
# with pairs of inflected form and its tag
def inflected_imperative_forms(form_VIAM):
    table_imperat = {'2SM': '',
                     '2SF': 'ِE',
                     '2DN': 'َE',
                     '2PM': 'ُEا',
                     '2PF': 'نَ'}

    cod = table_imperat.keys()

    infYtag_VIAM= STEM_plus_suf(form_VIAM, cod, table_imperat, 'VIAM')

    return (pass_list_to_dictionary(infYtag_VIAM))



# This function adds imperfective inflection to i-stems active and passive,
# without the suffixes for mood
def processing_imperfective(dict_forms):
    table_imperf_pre = {'1SN': 'أ',
                        '1PN': 'ن',
                        '2SM': 'ت',
                        '3SM': 'ي',
                        '3SF': 'ت'}
    cod_prefs = table_imperf_pre.keys()

    table_imperf_pre_suf = {'2SF': '_تِE',
                            '2DN': '_تَE',
                            '2PM': '_تُE',
                            '2PF': 'تـْنَ',
                            '3DM': '_يَE',
                            '3DF': '_تَE',
                            '3PM': '_يُE',
                            '3PF': 'يـْنَ'}
    cod_pref_sufs = table_imperf_pre_suf.keys()


    pre_Active = pre_plus_STEM(dict_forms['VI-A'], cod_prefs,
table_imperf_pre,'')+ \
                 pre_plus_STEM_plus_suf(dict_forms['VI-A'], cod_pref_sufs,
table_imperf_pre_suf,'')

    pre_Pasive = pre_plus_STEM(dict_forms['VI-P'], cod_prefs,
table_imperf_pre,'')+ \
```

```
                pre_plus_STEM_plus_suf(dict_forms['VI-P'], cod_pref_sufs,
table_imperf_pre_suf,'')


    return({'Active': pass_list_to_dictionary(pre_Active),
            'Pasive': pass_list_to_dictionary(pre_Pasive)})


# This function produced a dictionary with the imperfective forms for
passive or active i-stems
# The function gets as parameters a dictionary of imperfective forms, a
table with the rules to be applied,
# a flag f indicating if it is active or passive, and a flag t
(N:Indicative, S:Subjunctive, Y:Jussive)
def processing_Indic_Subj_Juss_AorP(forms_preprocess, table, f, t):
    keys=forms_preprocess.keys()

    ImperfParadigm={}

    for k in keys:
        if k in table:
            ImperfParadigm.setdefault('VI'+f+t+k,
forms_preprocess[k]+table[k])
        else:
            ImperfParadigm.setdefault('VI'+f+t+k, forms_preprocess[k])

    return (ImperfParadigm)


# This function adds indicative inflection to active and passive i-stems
# and includes them in a dictionary
def processing_Indicative(d_pre_proces):
    table_indicat = {'1SN': 'ُ',
                     '1PN': 'ُ',
                     '2SM': 'ُ',
                     '2SF': 'َ',
                     '2DN': 'ِ',
                     '2PM': 'َ',
                     '3SM': 'ُ',
                     '3SF': 'ُ',
                     '3DM': 'ِ',
                     '3DF': 'ِ',
                     '3PM': 'َ'}
    d_Indicative={}
```

```
    d_Indicative['Active']=
processing_Indic_Subj_Juss_AorP(d_pre_proces['Active'], table_indicat, 'A',
'N')
    d_Indicative['Pasive']=
processing_Indic_Subj_Juss_AorP(d_pre_proces['Pasive'], table_indicat, 'P',
'N')
    return (d_Indicative)

# This function adds subjunctive inflection to active and passive i-stems
# and includes them in a dictionary
def processing_Subjunctive(d_pre_proces):
    table_subjunctive = {'1SN': 'ो',
                         '1PN': 'ो',
                         '2SM': 'ो',
                         '2PM': 'ु',
                         '3SM': 'ो',
                         '3SF': 'ो',
                         '3PM': 'ु'}

    d_Subjunctive={}
    d_Subjunctive['Active']=
processing_Indic_Subj_Juss_AorP(d_pre_proces['Active'], table_subjunctive,
'A', 'S')
    d_Subjunctive['Pasive']=
processing_Indic_Subj_Juss_AorP(d_pre_proces['Pasive'], table_subjunctive,
'P', 'S')
    return (d_Subjunctive)


# This function adds jussive inflection to active and passive i-stems
# and includes them in a dictionary
def processing_Jussive(d_pre_proces):
    table_yusive = {'2PM': 'ु',
                    '3PM': 'ु'}

    d_Yusive={}
    d_Yusive['Active']=
processing_Indic_Subj_Juss_AorP(d_pre_proces['Active'], table_yusive, 'A',
'Y')
    d_Yusive['Pasive']=
processing_Indic_Subj_Juss_AorP(d_pre_proces['Pasive'], table_yusive, 'P',
'Y')
    return (d_Yusive)


# This function applies all the inflection to imperfctive forms
```

```
def inflected_imperfective_forms(dict_forms):
    d_imperf = processing_imperfective(dict_forms)
    d_Indicative = processing_Indicative(d_imperf)
    d_Subjunctive = processing_Subjunctive(d_imperf)
    d_Jussive = processing_Jussive(d_imperf)


    return({'Active': {'Indicative': d_Indicative['Active'],
                       'Subjunctive': d_Subjunctive['Active'],
                       'Yusive': d_Jussive['Active']},
            'Pasive': {'Indicative': d_Indicative['Pasive'],
                       'Subjunctive': d_Subjunctive['Pasive'],
                       'Yusive': d_Jussive['Pasive']},})




# this function passes each form to the full inflectional paradigm
# and creates a dict of forms
def Inflectional_system (dict_forms):
    dict_Inf_forms={}

    dict_Inf_forms['VP'] = inflected_perfective_forms(dict_forms)
    dict_Inf_forms['VI'] = inflected_imperfective_forms(dict_forms)
    dict_Inf_forms['VIAM'] = inflected_imperative_forms(dict_forms['VIAM'])


    return (dict_Inf_forms)
```

myre.py

```
import sys
import re
class myclase():
    def __init__(self):
        self.re=re
        self.cuenta={}
    def __getattr__(self, atr):
        return self.re.__dict__[atr]
    def sub(self,*a,**b):
        clave=a[0]+'::'+a[1]
        resultado=self.re.sub(*a,**b)
        if resultado!=a[2]:
```

```
            self.cuenta[clave]=self.cuenta.get(clave,0)+1
        return resultado

#print (myre.cuenta)
sys.modules[__name__] = myclase()
```

phonotactics.py

```
########################################################################
# MODULE   PHONOTACTIC CONSTRAINTS AND ORTHOGRAPHIC NORMALIZATION
########################################################################

from GenerationVerbs import myre as re
import sys


# This function sukunizes the form and applies long vowels normalization.
def Sukun_and_Long_vowels_normalization(form):

    # sukunizer
    form=re.sub(r"([[[ِ?))([ءأئؤبتثجحخدذرزسشصضطظعغفقكلمنوهي])|(?=$))",
r"\1ْ", form)
                                                                    # C1
sukunizer
    # long vowels normalization
    form = form.replace('ْE', 'اْ')                                 # C2
aE -> aE
    form = form.replace('ُE', 'وُ')                                 # C3
uE -> uw
    form = form.replace('ِE', 'يِ')                                 # C4
iE -> iy

    return(form)



# initial consonant cluster constrints rules - preprocessing initial
def  preprocessing_Initial_Alif(form):

    C = r'[بتثجخحدذرزسشصضطظعغفقكلمنهويأءؤئ]'
    form=re.sub(r'^({0}ْ{1ُ})'.format(C,C), r'1\أ',form)            # D1
Ø -> Au / ^_C·C~?u
```

```
    form=re.sub(r'^({0}ْ{1^]?}])'.format(C,C), r'1\ٕ',form)          # D2
Ø -> Ai / ^_C·C~?[^u]

    return form



def Apply_WeakLettersRules(form, al_code, lema, root, d_code,
verbList_apoc):  # added verbList_apoc

##    COMPR = (al_code == 'VPAN3SM' and lema == 'ابتاع' and root == 'بيع')
##    if COMPR: print('* '+form)

    # ------------------------- APOCOPATED IMPERATIVE ----------------------
---- #

    # A1 BY APPLYING A LIST FILTER

    # looks if the verb has apocopated imperative from the list
verbList_apoc and if so removed the first part of the form
    # all verbs in the list are going to be simple verbs
    codeString=d_code['Internal derivation']['lengthening']+\
               d_code['Internal
derivation']['addition']+d_code['Template']+\
               d_code['External derivation']+d_code['Vocalization']['Perf
V2']+\
               d_code['Vocalization']['Imperf
V1']+d_code['Vocalization']['Imperf V2']
    if al_code[:4]=='VIAM' and ((lema,root,codeString) in verbList_apoc):
        form=form[4:]                                                  # E1

    # A1 BY APPLYING A RULE FILTER

    # if simple verb, imperative form, first radical waw; and if (thematic
vowel of imperfective kasra) or
    # ((second or third radical is a guttural ه ء ع ح غ خ or semiguttural ر)
and (thematic verb is fatha)):
    # then the fist radical is removed from the form (plus prosthetic alif)
##    Gutturals = ['ر','ه','ء','ع','ح','غ','خ']
##    if d_code['Internal derivation']['lengthening']=='0' and
d_code['Internal derivation']['addition']=='0'\
##       and d_code['Template']=='L' and al_code[:4]=='VIAM' and
root[0]=='و'\
##       and ((d_code['Vocalization']['Imperf V2']=='0') or
((d_code['Vocalization']['Imperf V2']=='2')\
```

```
##                                                          and (root[1] in
Gutturals or root[2] in Gutturals))):
##            form=form[4:]                                                 # E1
    # ------------------------------------------------------------------------
----- #

    form=re.sub('^ئِ','إِ',form)                                            # E2
y -> w / ^Ai_·    # iw constraint for imperatives

    if (d_code['Internal derivation']['addition']!='3')and
(re.search('^[وي]',root))and(len(root)!= 4):
        form=re.sub('^([]ي-ء])ُ()َ-ء)',r'\1\2',form)                        # E3
w· -> Ø / ^Ca_Ci

    form=re.sub('(.[])وْ](؟)َ-ء(?!$)',r'\1ي',form)                          # E4
[wy]· -> y / [^^]C~?i_[^$]

    form=re.sub('([])وْ]([ِ-[؟]َ-ء$',r'\1',form)                            # E5
[wy]· -> Ø / C~?v_$

    if d_code['Internal derivation']['lengthening']!='1' and not
re.match('[12]',d_code['Internal derivation']['addition'])\
       and len(root)!=4 and re.search('^.(?![وي])', lema):

        form=re.sub('([]ْ-ء]=?)([ِ-(([[وي]])ْ-ء)',r'\1\2',form)             # E6
·[wy] -> Ø / C_vC·

        form=re.sub('([ِ-[ٔ)(ي!?)]وَ]])ْ-ء])',r'\12\ا◌ٔ',form)              # E7
·[wy] -> A / C_a[^y]

        form=re.sub('(?<=[ِ-[[ي^])]وي]])ْ-ء])',r'1\يِ',form)               # E8
·[wy]i -> iy / C_[^y]v

        form=re.sub('([ِ-[[ي^])((ٖ[وي]])ْ-ء])',r'\1\23\و',form)            # E9
·[wy]u -> uw / C_[^y]v

        form=re.sub('^([ِ-[[ي^])[ِ-[[وي](]َ-ء])',r'\12\ا',form)            # E10
[wy]v -> A / ^Ca_[^y]v

        form=re.sub('(.[]َ-ء])]وَ](؟)َ-ء)',r'\1\2',form)                    # E11
[wy]a -> Ø / .C~?a_Ca

    form=re.sub('.)ُو)',r'1\يِ',form)                                       # E12
uwi -> iy / _Ca
```

```
    if re.match('[124]',d_code['Internal derivation']['addition']):
# cambiada
        form=re.sub('([ِ-ٕ[^ي])[ِ-ٕ[وي](]ىَ-ء])',r'\12\ا',form)       # E13
adaptada para VII-VIII-X  [wy]v -> A / Ca_[^y]v

    if (d_code['Vocalization']['Imperf
V2']=='1')and(re.search('[ي!?)(و!?)[وي).$',root)):
        form=re.sub('([]ْ-ء])وَ])ىَ-ء)',r'\1ُ\2',form)                  # E14
awa -> u / C_C·

    if (d_code['Vocalization']['Imperf V2']!='1') and len(root)==3 and not\
       re.match('[12]',d_code['Internal derivation']['addition']):
form=re.sub('([]ْ-ء])[[[وي]])ىَ-ء.)',r'\1ِ\2',form)          # E15   a[wy]a -
> i / C_C·.

    if (re.search('^.[^وي][وي]$', root) or re.search('^.[وي^][وي]$', root)):
## antigua if (re.search('[وي]$', root)):
        form=re.sub('(ً)[]ْ-ء])[[[وي)',r'\1\2',form)                  # E16
[wy][ai] -> Ø / a_C·

    form=re.sub('[ُِ][وَ(ٰوي)',r'\1',form)                             # E17
[ui][wy] -> Ø / _uW

    if d_code['Vocalization']['Perf V2']!='2' or (root[1]!='و' and
root[1:]!='يي'):
        form=re.sub('ُ[ْ(وي)',r'\1',form)                              # E18
u[wy] -> Ø / C_iC·   ##NUEVA

    if ((d_code['Vocalization']['Perf
V2']!='1')|(d_code['Vocalization']['Imperf V2']!='1')):
        form=re.sub('([ي(وي][[ٰ?]ىَ-ء)',r'\1\2',form)               # E19
[ui][wy] -> Ø / C~?_iy

    form=re.sub('(ً)[وُ]وُ',r'\1ٗ',form)                              # E20
[wy]uw -> w· / a_

    if (d_code['Vocalization']['Imperf V2']!='1') or al_code[2]=='P':
# pasiva & imperf en u sí cambia a alif maqsura
        form=re.sub('([^[ٰوي](]ىَ]$',r'\1ى',form)                     # E21
[wy][au]  -> Y / [^y]a_$

    if (d_code['Vocalization']['Imperf V2']=='1'):
        form=re.sub('(ً)وَ$',r'\1ا',form)                               #
E22   wa -> A / a_$
```

```
    form=re.sub('(يَ(يَ$',r'\1ا',form)                                    #
E23   ya -> A / ya_$

    form=re.sub('()وُ$',r'\1',form)                                      # E24
u -> Ø / uw_$

    form=re.sub('وِ',r'يِ',form)                                          #
E25   w -> y / i_a    # segundo retoque, REGLA MUY GENERALIZADA!!

    if(al_code=='VPAN3SF'):
        form=re.sub('([]ْ-ء])]وَ](]يَ-ء)$',r'\1\2',form)                 # E26
[wy]a -> Ø / Ca_C·$

    form=re.sub('()ْوا',r'\1',form)                                      # E27
· -> Ø / (uw|iy)_

    form=re.sub('(ِ)[]وُ$',r'\1ي',form)                                   # E28
[wy]u -> y / i_$

    if re.search('^..و$',root) and\
       ((d_code['Internal derivation']['lengthening']!='0' or
d_code['Internal derivation']['addition']!='0')\
       and not re.match('V[IP][AP][NSY]3PM',al_code) and not
re.match('VI[AP][NSYM]2PM',al_code))\
        or ((d_code['Internal derivation']['lengthening']=='0' and
d_code['Internal derivation']['addition']=='0')\
        and (d_code['Vocalization']['Perf V2']=='0' and
d_code['Vocalization']['Imperf V2']=='1')\
        and (re.match('VIP[NSY][23]D[FMN]',al_code) or
re.match('VIP[NSY][23]PF',al_code) or re.match('VIPN2SF',al_code))):

        form=re.sub('([و(?]يَ-ء',r'\1ي',form)                             # E29
w -> y / C~?a_

    form=re.sub('ْ','ِيـَ',form)                                          # E30
yiy -> y· / a_

    return(form)



def Apply_ShaddaRules(form, al_code, lema, root, d_code):

    if (len(root)==4 and d_code['Internal derivation']['lengthening']=='1')\
       or (len(root)==3\
```

```
            and ((not 'ت' in root)\
            or (re.search('[^^]ـتـ[]ت',form))\
            or (root[0]=='ت' and (d_code['External derivation']!='1') and not
(d_code['Internal derivation']['addition']=='4' and root[:2]=='تـو'))\
            or (re.match('^.تـت$',root) and not re.match('VP[AP]N3DF',al_code)
and d_code['Internal derivation']['addition']!='2'))):

             form=re.sub(r'([^ْ])([ِ-([2\[ِ-[([ي-ء])',r'\1\2ّ\3',form) # F1
C¹v¹C¹ -> C¹~ / [^·]_v² (and not beginning)
             form=re.sub(r'([ِ-([2\([ِ-([[([ي-ء])]ْ)ي-ء])',r'\1\3\2ّ\4',form) #
C2 ·C²v¹C² -> v¹C²~ / C¹_v²

    if root[1]==root[2] and not re.match('[YM]',al_code[3]):
        form=re.sub(r'ْ([1ّ\([ِ-([([ي-ء$',r'\2\1ّْ',form)              # F3
·C¹v¹C¹· -> v¹C¹~a / _$

    # ORTHOGRAPHIC
    form=re.sub(r'([ِ-([1\])ْ-ء])',r'\1ّ\2',form)                     # F4
·C¹ -> ~ / C¹_v¹     (C¹·C¹v¹ -> C¹~v¹)

    return(form)



# initial consonant cluster constrints rule - posprocessing initial
def  posprocessing_Initial_Alif(form):

    C = r'[بـتـثجحخدذرزسشصضطظعغفقكلمنهويـأءؤئ]'
    form=re.sub(r'^[{0})[[أ])'.format(C), r'\1',form)             # D3   Av
-> Ø / ^_Cv

    return form



def Apply_HamzaRules(form):

    try:

        form_trasl=form

        form_trasl=re.sub('^[إ','][ـئـأءؤ',form_trasl)                    # G1
[cúýÁ] -> À / ^_i  eg إبـدَاء

        form_trasl=re.sub('^[أ','([[ـّ?)[ـئـبءؤ',form_trasl)             # G2
[cúýÀ] -> Á / ^_[au]  eg أَسرَةْ
```

```
        form_trasl=re.sub('(^|[^ا][ءأإؤئ])([ِيُ]','\1آ',form_trasl)       # G3
[cÁÀúý]aA -> Ã / (^|[^Auiy])_  eg قُـرآن / آسِف

        form_trasl=re.sub('[آ',']ءأإؤئ][ءأإؤئ',form_trasl)            # G4
[cÁÀúý]a[cÁÀúý]· -> Ã  eg آتـكِلُ

        form_trasl=re.sub('ْ[أ',']ءإؤئ',form_trasl)                  # G5
[cÀúý] -> Á / ·_a  eg أبـأرُ

        form_trasl=re.sub('ْ[ؤ',']ءأإئ',form_trasl)                  # G6
[cÁÀý] -> ú / ·_u  eg أبـؤس

        form_trasl=re.sub('ا[([[أإؤئ]ي?)$',r'1\ء',form_trasl)    # G8
[ÁÀúý] -> c / y_[aiuâîû]A?$  eg جُزَيءُ

        form_trasl=re.sub('يـئ','(..=?)(ا!?)[ءأإؤ]ي',form_trasl)        # G9
[cÁÀú] -> ý / y_(?!âA)(?=..)  eg بَـدِيـئَة

        form_trasl=re.sub('ءا',']أإؤئ]ا',form_trasl)                  # G10
[ÁÀúý] -> c / A_a  eg بَـدَاءَة

        form_trasl=re.sub('ُ[ؤ','([[ً?)[ءأإئ',form_trasl)           # G11
[cÁÀý] -> ú / u_[a·u]  eg بَـطُؤتُ

        form_trasl=re.sub('وؤ','(ا!?)(.[[ً?)[ءأإئ]و',form_trasl)     # G12
[cÁÀý] -> ú / w_(?=[au].)(?!âA)  eg مَـوبُـوؤيـن

        form_trasl=re.sub('ا[([[أإؤئ]و?)$',r'1\ء',form_trasl)      # G13
[ÁÀúý] -> c / w_[auâû]A?$  eg بُـرُوءُ

        form_trasl=re.sub('َ[أ',']([[?=?)[ءإؤئ',form_trasl)          # G14
[cÀúý] -> Á / a_~?[a·]  eg وَثـأتُ

        form_trasl=re.sub('(?<=[ؤ','(.=?)]ءأإئ]([اَ',form_trasl)       # G15
[cÁÀý] -> ú / [aA]_u. eg يُـوثـؤُوا

        form_trasl=re.sub('َ[([[ءإؤئ])$',r'1\أَ',form_trasl)          # G16
[cÀúý] -> Á / a_[uû]$  eg وَثـأُ

        form_trasl=re.sub('([[أإؤئ]ا])$',r'1\ء',form_trasl)          # G17
[ÁÀúý] -> c / A_[uû]$  eg وَثـاءُ

       form_trasl=re.sub('(?<!ُ)[؟ؤئ]ءأإ)$',r'1\إ',form_trasl)  # D18 [cÁúý]
-> À / [^·]_~?i$  eg يَـنـتَإ (V) / نَـبَإ (N)
```

```
        form_trasl=re.sub('([^ًٌ])[?ؤ(ئ|ء|أ|إ|ؤ)$',r'\12\ئ',form_trasl)    # G18a
[cÁÀú] -> ý / [^·a]_~?i$
        form_trasl=re.sub('ً[]ؤ|ء|أ|إ$',r'ئً',form_trasl)                  #
G18b [cÁÀú] -> ý / a_i$  # NUEVA !! para a sin shadda
        form_trasl=re.sub('ً[]ؤ|ئـ|ـئ|ء$',r'ئً',form_trasl)                # G18b
[cýÀú] -> Á / a_~i$  # NUEVA !! para a con shadda


        form_trasl=re.sub('(?<=.)[ئ','(.?=?)[ؤ|ء|أ|إ',form_trasl)        # G19
[cÁÀú] -> ý / ._~?i[^$]  eg ئـنـئِيَـا

        form_trasl=re.sub('ْ[ء','$]ئ|ؤ|أ|إ',form_trasl)  # D20 [ÁÀúý] -> c /
·_i$  eg وَطِء (N) / أَصِئ (V)
        form_trasl=re.sub('ْ[إ','$]ئ|ؤ|أ|ء',form_trasl)                  # G20
[cÁúý] -> À / ·_i$  eg أَصِئ

        form_trasl=re.sub('ِ[ئ','[ؤ|إ|أ|ء',form_trasl)                    # DG7
[cÁÀú] -> ý / i_ eg جَـآجِئُ

        return(form_trasl)

    except ValueError as verror:
        print('Value error in function Apply_RulesPostTrasl_form(): ' +
str(verror))


# This function receives as paremeters a list with a dictionary containing
the verbal forms and CODE, lema, root, d_code
# and applyies the phonotactic and orthographic rules returning a list with
the surface forms and their CODE.
def Irreg_rules_list(d_form, lema, root, d_code):
    keys=d_form.keys()

    # --------------- take verbs with apocopated imperative ----------------
-------------
    ApocopVerbs=[]
    with
open('GenerationVerbs\Apocopated_Imperative_verbs.txt',encoding='utf8') as
file_imper:
        for each_verb in file_imper:
            try: l_imperat,r_imperat,c_imperat=each_verb.strip().split()
            except: print('error in Apocopated_Imperative_verbs file, line:
%s' % each_verb)
            ApocopVerbs.append((l_imperat,r_imperat,c_imperat))
    # ------------------------------------------------------------------------
-------------
```

```
    d_norm_f={}
    for k in keys:
        # Sukun and long vowel rules
        form = Sukun_and_Long_vowels_normalization(d_form[k])
        # initial consonant clusters preprocessing
        form = preprocessing_Initial_Alif(form)
        # Weak letters
        form = Apply_WeakLettersRules(form, k, lema, root, d_code,
ApocopVerbs) # added verbList_apoc
        # Shadda rules
        form = Apply_ShaddaRules(form, k, lema, root, d_code)
        # initial consonant clusters posprocessing
        form = posprocessing_Initial_Alif(form)
        # Hamza rules
        form = Apply_HamzaRules(form)
        d_norm_f.setdefault(k, form)

    return d_norm_f


# This function receives as paremeters a dictionary with the verbal form,
lema, root, d_code
# and applies the phonotactic and orthographic rules returning a dictionary
with the surface forms
def phonotactic_rules(dict_form, lema, root, d_code):

    dict_form['VP']= {'Active': Irreg_rules_list(dict_form['VP']['Active'],
lema, root, d_code),
                      'Pasive': Irreg_rules_list(dict_form['VP']['Pasive'],
lema, root, d_code)}
    dict_form['VI']['Active']= {'Indicative':
Irreg_rules_list(dict_form['VI']['Active']['Indicative'], lema, root,
d_code),
                                'Subjunctive':
Irreg_rules_list(dict_form['VI']['Active']['Subjunctive'], lema, root,
d_code),
                                'Yusive':
Irreg_rules_list(dict_form['VI']['Active']['Yusive'], lema, root, d_code)}
    dict_form['VI']['Pasive']= {'Indicative':
Irreg_rules_list(dict_form['VI']['Pasive']['Indicative'], lema, root,
d_code),
                                'Subjunctive':
Irreg_rules_list(dict_form['VI']['Pasive']['Subjunctive'], lema, root,
d_code),
```

```
                                'Yusive':
Irreg_rules_list(dict_form['VI']['Pasive']['Yusive'], lema, root, d_code)}
    dict_form['VIAM'] = Irreg_rules_list(dict_form['VIAM'], lema, root,
d_code)

    return(dict_form)
```

verbs_tester.py

```
import re

from GenerationVerbs import GV

##import GV


# TESTER FOR VERBS
def test_verbs():
    with open('ResultadosFile\lexiconVerbs_jabalin_test.txt',
encoding='utf8') as VerbsFile,\
         open('GenerationVerbs\jabalin_verbs_test.txt', encoding='utf8') as
TestFile:

        pasados, fallados = 0, 0
        Test, Verbs = {}, {}

        for line in TestFile:
            line = re.sub('(#.+)','',line)
            if re.search('\S',line):
                try: f, f_c, l, r, l_c = line.strip().split('\t',4)
                except: print('Error %s' % line)
                Test[(f_c, l, r, l_c)] = f


        for line in VerbsFile:
            try: form, form_code, lema, root, lema_code =
line.strip().split('\t',4)
            except: print('Error %s' % line)
            Verbs[(form_code, lema, root, lema_code)] = form


        for info,word in Verbs.items():
```

```
            for info_t,word_t in Test.items():

                if (info_t == info):

                    if  word_t == word:
                        print('OK\t{}\t{}  {} {}
{}'.format(word_t,info_t[0],info_t[1],info_t[2],info_t[3]))
                        pasados+=1
                    else:
                        print('\nFAIL!\tGenerated form:\t{} {} {} {}
{}'.format(word,info[0],info[1],info[2],info[3]))
                        print('\tCorrect form:\t{}\t{}  {} {}
{}\n'.format(word_t,info_t[0],info_t[1],info_t[2],info_t[3]))
                        with open('GenerationVerbs\jabalin_temp.txt', 'w',
encoding='utf8') as temp:
                            print(info[1]+'\t'+info[2]+'\t'+info[3],
file=temp)

GV.generation_verbs('GenerationVerbs\jabalin_temp.txt',
"GenerationVerbs\jabalin_temp_out.txt")
                        fallados+=1

        print('\n')
        for i in Test:
            if i not in Verbs:
                print('Info not found in lexicon:\t{} {} {}
{}'.format(i[0],i[1],i[2],i[3]))


        print (20*'--')
        print ('\n%s Tests\n' % len(Test))
        print ('passed: %s' % pasados)
        print ('failed: %s' % fallados)
```

utilities.py

```
########################################################################
# MODULE             UTILITIES
########################################################################
import re
import sys

# This function inverts a string
```

```
def invertir(var):
    return var[::-1]


# This function returns 'true' if the line passed as a parameter begins with
the word 'ENTRY'
# or if it's an empty line, else return 'false'
def delete_line(line):
    if re.match("ENTRY(.+)", line):
        return (True)
    if re.match("^\n", line):
        return(True)
    return(False)


# This function gets a code and returns a dictionary with its decodification
# or empty dictionary if something worked wrong
# i.e. {'Vocalization': {'Perf V2': '0', 'Imperf V2': '0', 'Imperf V1':
'0'},
#       'Internal derivation': {'lengthening': '1', 'addition': '0'},
#       'Template': 'H',
#       'External derivation': '0'}
def parse_code_verbs(code):
    if len(code)!=7:
        print("error CODE: " + code)
        return ()

    dict_parse={}
    dict_parse["Internal derivation"]={'lengthening': code[0], 'addition':
code[1]}
    dict_parse["Template"]=code[2]
    dict_parse["External derivation"]=code[3]
    dict_parse["Vocalization"]={'Perf V2': code[4], 'Imperf V1': code[5],
'Imperf V2': code[6]}

    return(dict_parse)


def printFile_forms(lema, root, code, dict_forms, n_file):
    k_forms = dict_forms.keys()

    for k in k_forms:
        print(dict_forms[k] + '\t'+ k + '\t' + str(lema) + '\t'+\
              str(root)+'\t'+ str(code), file=n_file)

def printFile_forms_from_dictForms(lema, root, code, d_forms, n_file):
```

```
    printFile_forms(lema, root, code, d_forms['VP']['Active'], n_file)
    printFile_forms(lema, root, code, d_forms['VP']['Pasive'], n_file)
    printFile_forms(lema, root, code, d_forms['VI']['Active']['Indicative'],
n_file)
    printFile_forms(lema, root, code,
d_forms['VI']['Active']['Subjunctive'], n_file)
    printFile_forms(lema, root, code, d_forms['VI']['Active']['Yusive'],
n_file)
    printFile_forms(lema, root, code, d_forms['VI']['Pasive']['Indicative'],
n_file)
    printFile_forms(lema, root, code,
d_forms['VI']['Pasive']['Subjunctive'], n_file)
    printFile_forms(lema, root, code, d_forms['VI']['Pasive']['Yusive'],
n_file)
    printFile_forms(lema, root, code, d_forms['VIAM'], n_file)
```

limpia_lema_freq.py

```
##import utilities

from GenerationVerbs import utilities

def limpiar_lema_freq():
    with open('GenerationVerbs\lexicon_lemas_freq_ini.txt', encoding='utf8')
as file_in, open('ResultadosFile\lexicon_lemas_freq.txt', 'w',
encoding='utf8') as file_out:
        for each_line in file_in:
            if not utilities.delete_line(each_line):
                try:
                    lema, root, code, freq = each_line.strip().split('\t',3)
                    print(lema+'\t'+root+'\t'+code, file=file_out)
                except:
                    print(each_line, file=file_out)
```

servidor_apli.py

```
#! /usr/bin/env python3
import subprocess
```

```
import webbrowser
import os

def arrancar():
    p = subprocess.Popen(['python.exe','simple_httpd.py'],
                         stdin=subprocess.PIPE,
                         stdout=subprocess.PIPE,
                         stderr=subprocess.PIPE)

    l=os.getenv('PROGRAMFILES')

    ffcommand = "c:/"+l[3:]+"/mozilla firefox/firefox.exe %s &"
    webbrowser.get(ffcommand).open_new("localhost:8080")
```

simple_httpd.py

```
from http.server import HTTPServer, CGIHTTPRequestHandler

port = 8080

httpd = HTTPServer(('', port), CGIHTTPRequestHandler)
print("Starting simple_httpd on port: " + str(httpd.server_port))
httpd.serve_forever()
```

eval.py

```
import sqlite3

import re
import glob

# This function returns a dictionary with clave conjCodeAlicia and  valor
conjCodeElixir
def dict_conjCodeAli_to_Elixir():
     with open('eval_elixir\etiquetas_verbos.txt') as file:
          d_conjCode={}
          for each_line in file:
               (conjCodeEli, conjCodeAli)=each_line.strip().split('\t',1)
               d_conjCode.setdefault(conjCodeAli, conjCodeEli)
```

```
          return (d_conjCode)


# This function printers a dictionary
def imprimedic(d):
     ks=d.keys()
     for k in ks:
          print(k + ':' + '   '+str(d[k]))

# This function returns a dictionary with the verb code of Alicia and its
patterns of elixir
def dict_verbCode_to_pattern():
     with open('elixir_patterns_to_our_code.txt') as file:
          d_verbCodeToPattern={}
          for each_line in file:
               resultado=each_line.strip().split('\t')
               pattern=resultado.pop(0)
               verbCodes=resultado
               for vC in verbCodes:
                    if vC in d_verbCodeToPattern:
                         if isinstance(d_verbCodeToPattern[vC], list):
                              d_verbCodeToPattern[vC].append(pattern)
                         else:

d_verbCodeToPattern[vC]=[d_verbCodeToPattern[vC], pattern]
                    else:
                         d_verbCodeToPattern.setdefault(vC, pattern)
          #print(imprimedic(d_verbCodeToPattern))
          return (d_verbCodeToPattern)


# This function returns the id, or NULL,  of the column which has form,
etiqueta and coordenade passed.
def busca_indiceII(form, etiqueta, coord, cursor):
     result=cursor.execute("""SELECT id FROM lexiconVerbs WHERE form=? AND
etiqueta=? AND coord=?""",\
                           (form,etiqueta,coord))
     id_result = result.fetchone()
     return (id_result)


# This function returns a list of the rows to evaluate with the forms to
evaluate.
def forms_to_eval(cursor):
```

```
     results=cursor.execute("""SELECT form, etiqueta, coord, cod_lema, lema,
root FROM lexiconVerbs WHERE eval='SI'""")
     results_eval = results.fetchall()
     return (results_eval)

# This function returns 'SI' si la forma es evaluable y 'NO' si no lo es
def form_to_eval(form, etiqueta, cod_lema, lema, root, cursor):
     results=cursor.execute("""SELECT eval FROM lexiconVerbs WHERE form=?
AND etiqueta=? AND cod_lema=? AND lema=? AND root=? AND eval='SI'""",\
                            (form, etiqueta, cod_lema, lema, root))
     results_eval = results.fetchone()
     if results_eval:
          return(results_eval[0])
     return(results_eval)



# This function returns a result with form of the column which has etiqueta
and coord passed.
def busca_indiceIIparcial(etiqueta, coord, cursor):
     result=cursor.execute("""SELECT form FROM lexiconVerbs WHERE etiqueta=?
AND coord=?""",\
                            (etiqueta, coord))
     r = result.fetchone()
     return(r)

def find_pattern(lpatternElix, lpatternAli):
     for e in lpatternElix:
          if e in lpatternAli:
               return('YES')
          else:
               return('NO')

def create_table_eval(cursor,nom_table):
     cursor.execute("CREATE TABLE "+ nom_table+" (id INTEGER PRIMARY KEY
AUTOINCREMENT UNIQUE NOT NULL,"\
                    'elixir_form TEXT,'\
                    'form TEXT NOT NULL,'\
                    'lema TEXT NOT NULL,'\
                    'root TEXT NOT NULL,'\
                    'cod_lema TEXT NOT NULL,'\
                    'etiqueta TEXT NOT NULL,'\
                    'coord TEXT,'\
                    'RDO TEXT)')
```

```
def add_eval_in_table(elixir_form, form, etiqueta, cod_lema, lema, root,
cursor, nom_table, coord='', rdo=''):
     cursor.execute('INSERT INTO '+nom_table+"(elixir_form, form, etiqueta,
cod_lema, lema, root, coord,RDO) VALUES(?, ?, ?, ?, ?, ?, ?, ?)",\
                    (elixir_form, form, etiqueta, cod_lema, lema, root,
coord, rdo))

def num_rowls_table(cursor):
     r=cursor.execute("""SELECT COUNT(*) FROM table_fail""")
     r=r.fetchone()
     return(r[0])

### This function removes the forms which have been generated of different
forms
##def analizar(conj_code, root, verbs_code):
##     not_root='ء'
##     l_not_verb_code=['20H0010','30H0010','03H0010','00H0010']
##
##     # forms removed: imperfective, passive, root beginnign with hamza eg.
أُوبَـرُ / أُؤْبَـرُ
##     if (not_root==root[0]) and (conj_code[1]=='I') and
(conj_code[2]=='P'):
##          return('NO')
##     # forms removed: imperfective, active, forms II III IV and I-quad,
root beginnign with hamza
##     elif (verbs_code in l_not_verb_code) and (not_root==root[0]) and
(conj_code[1]=='I') and (conj_code[2]=='A'):
##          return('NO')
##     return('SI')

# This function said if a form is not evaluable because, e.g, it has been
generated the different way. Return 'NO' if it's not evaluable,
# and 'SI' in other case
def evaluable(etiqueta, root, cod_lema):
     # I-IV, VIII, QI
     l_codes_lema_not_eval=['00L0001', '00L0000', '00L0002', '00L0101',
'00L0202', '00L0200', '20H0010', '30H0010', '02L0000', '00H0010']
     # Primer caso:
     if (re.search('VIP[NSY]2SF', etiqueta) and
((root[2]=='و')or(root[2]=='ي')) and len(root)==3):
          return('NO')

     if (root[0]=='ء'):
          if (re.search('VIP[NSY]1SN', etiqueta)) and (cod_lema in
l_codes_lema_not_eval):
               return('NO')
```

```
          if (re.search('VIA[NSY]1SN', etiqueta)) and (cod_lema ==
'03H0010'):
               return('NO')

          if (re.search('V[IP]P[NSY]', etiqueta)) and (cod_lema ==
'03H0010'):
               return('NO')

     return('SI')

def evaluacion():
     db_name_elixir='ResultadosBdD\lexiconVerbsdata_Elixir.sqlite'
     db_name_jabalin='ResultadosBdD\lexiconVerbsdata_Jabalin.sqlite'

     dict_etiqJabalin_to_etiqElixir = dict_conjCodeAli_to_Elixir()

     num_accert=0
     num_fail_real=0
     num_fail=0
     num_not_evaluable=0

     connection_elixir = sqlite3.connect(db_name_elixir)
     cursor_elixir=connection_elixir.cursor()
     connection_jabalin = sqlite3.connect(db_name_jabalin)
     cursor_jabalin = connection_jabalin.cursor()


     create_table_eval(cursor_jabalin, 'table_fail_bin')
     create_table_eval(cursor_jabalin, 'table_fail_real')
     create_table_eval(cursor_jabalin, 'table_correct')
     create_table_eval(cursor_jabalin, 'table_not_evaluable')

     # recopilamos las formas que deben ser evaluadas

     forms_to_evaluate= forms_to_eval(cursor_jabalin)


     # 1er matching: form+etiqueta+coord. At last it generates tables
corrects
     # with enters haven't been found
     for f in forms_to_evaluate:
          form = f[0]
          etiqueta_jabalin = f[1]
          etiqueta = dict_etiqJabalin_to_etiqElixir[f[1]]
          coord_f = f[2]
```

```
          cod_lema = f[3]
          lema = f[4]
          root = f[5]

          id_r= busca_indiceII(form, etiqueta, coord_f, cursor_elixir)
          if id_r:
               form_elixir=form
               num_accert+=1
               nom_table = 'table_correct'


          # looking for not evaluation. If it is not evaluable then we put
in the table_not_evaluable
          elif evaluable(etiqueta_jabalin, root, cod_lema) == 'NO':
               num_not_evaluable+=1
               form_elixir = form_elixir[0]
               nom_table= 'table_not_evaluable'
          # 2º matching: tiqueta+coord
          else:
               form_elixir = busca_indiceIIparcial(etiqueta, coord_f,
cursor_elixir)
               if (form_elixir):
                    form_elixir = form_elixir[0]
                    num_fail+=1
                    num_fail_real+=1
                    nom_table = 'table_fail_real'
               else:
                    form_elixir=form
                    num_fail+=1
                    nom_table = 'table_fail_bin'

          add_eval_in_table(form_elixir, form, etiqueta, cod_lema, lema,
root, cursor_jabalin, nom_table, coord=coord_f)



     print('_________________EVALUATION_________________')
     print('NÚMERO DE ACIERTOS:    ' + str(num_accert))
     print('NÚMERO DE FALLOS TOTALES:      ' + str(num_fail))
     print('NÚMERO DE FALLOS REALES:      ' + str(num_fail_real))
     print('NÚMERO DE FORMAS NO EVALUABLES:      ' + str(num_not_evaluable))
     print('____________________________________________')


     connection_jabalin.commit()
     connection_jabalin.close()
```

```
     connection_elixir.commit()
     connection_elixir.close()


# This function returns 'id' if the form is in the table passed as a
parametre, if not returns 'NULL'
def encontrar_form_table(form, etiqueta, lema, root, cod_lema, cursor,
nom_table):
     result=cursor.execute('SELECT id FROM '+nom_table+" WHERE form=? AND
etiqueta=? AND lema=? AND root=? AND cod_lema=?",\
                           (form, etiqueta, lema, root, cod_lema))
     result = result.fetchone()
     return (result)



# This function returns a new table in BdDJabalin with the localitation of
the verbs more frequenty.
def Vbs_mas_freq():

     db_name_jabalin='ResultadosBdD\lexiconVerbsdata_Jabalin.sqlite'
     dict_etiqJabalin_to_etiqElixir = dict_conjCodeAli_to_Elixir()


     num_NO_EVAL=0
     num_NO_FOUND=0
     num_CORRECT=0
     num_FAIL=0
     num_DIF_CONJ=0
     num_verb=0

     connection_jabalin = sqlite3.connect(db_name_jabalin)
     cursor= connection_jabalin.cursor()


     create_table_eval(cursor, 'table_freq')

     with open('ResultadosFile\lexiconVerbs_freq.txt', encoding='utf8') as
file_freq:
          for f in file_freq:
               num_verb+=1
##               print(num_verb)
               form, etiqueta, lema, root, cod_lema = f.strip().split('\t',
4)
               etiqueta_elixir = dict_etiqJabalin_to_etiqElixir[etiqueta]
```

```
                rdo_lexicon_Verbs = encontrar_form_table(form, etiqueta,
lema, root, cod_lema, cursor, 'lexiconVerbs')
##                print('Lexicon_verbs')
##                print(rdo_lexicon_Verbs)
                # está en el lexicón completo de Verbos Jabalin
                if rdo_lexicon_Verbs:
                     evaluado=form_to_eval(form, etiqueta, cod_lema, lema,
root, cursor)
##                     print('evaluar')
##                     print(evaluado)
                     # Está y es evaluable
                     if evaluado=='SI':
                          rdo_table_correct = encontrar_form_table(form,
etiqueta_elixir, lema, root, cod_lema, cursor, 'table_correct')
##                           print('table correct')
##                           print(rdo_table_correct)
                          # Si está, es evaluable y está en correctos
                          if rdo_table_correct:
                               RDO='CORRECT'
##                               print('CORRECT')
                               num_CORRECT+=1
                          # Si está, es evaluable y no está en correctos
                          else:
                               rdo_table_fail = encontrar_form_table(form,
etiqueta_elixir, lema, root, cod_lema, cursor, 'table_fail_real')
##                                 print('table_fail')
##                                 print(rdo_table_fail)
                               # Si está, es evaluable pero es un fallo real
                               if rdo_table_fail:
                                    RDO='FAIL'
##                                     print('FAIL')
                                    num_FAIL+=1
                               # Si está, es eavluable y hay una diferencia
de conjugación
                               else:
                                    RDO='DIF_CONJ'
##                                     print('DIF_CONJ')
                                    num_DIF_CONJ+=1
                     # Está pero no es evaluable
                     else:
##                            print('NO_EVAL')
                          RDO='NO_EVAL'
                          num_NO_EVAL+=1
                # No está en el lexicón completo de Verbos Jabalín
                else:
```

```
                    RDO='NO_FOUND'
##                    print('NO_FOUND')
                    num_NO_FOUND+=1

                add_eval_in_table('NOT_WORK', form, etiqueta, cod_lema, lema,
root, cursor, 'table_freq', rdo=RDO)

     print('_________________EVALUATION VERBOS MÁS FRECUENTES_________________')
     print('NÚMERO DE NO_FOUND:      ' + str(num_NO_FOUND))
     print('NÚMERO DE NO_EVAL:    ' + str(num_NO_EVAL))
     print('NÚMERO DE CORRRECT:      ' + str(num_CORRECT))
     print('NÚMERO DE FAIL:      ' + str(num_FAIL))
     print('NÚMERO DE DIF_CONJ:      ' + str(num_DIF_CONJ))
     print('____________________________________________________________')


     connection_jabalin.commit()
     connection_jabalin.close()

# This function returns a new table in BdDJabalin with the localitation of
the verbs more frequenty.
def Vbs_mas_freq_v2():

     db_name_jabalin='ResultadosBdD\lexiconVerbsdata_Jabalin.sqlite'
     dict_etiqJabalin_to_etiqElixir = dict_conjCodeAli_to_Elixir()


     num_NO_EVAL=0
     num_NO_FOUND=0
     num_CORRECT=0
     num_FAIL=0
     num_DIF_CONJ=0
     num_verb=0

     connection_jabalin = sqlite3.connect(db_name_jabalin)
     cursor= connection_jabalin.cursor()


     create_table_eval(cursor, 'table_freq')

     with open('ResultadosFile\lexiconVerbs_freq.txt', encoding='utf8') as
file_freq:
          for f in file_freq:
               num_verb+=1
               print(num_verb)
```

```
                form, etiqueta, lema, root, cod_lema = f.strip().split('\t',
4)
                etiqueta_elixir = dict_etiqJabalin_to_etiqElixir[etiqueta]


                rdo_table_not_evaluable = encontrar_form_table(form,
etiqueta, lema, root, cod_lema, cursor, 'table_not_evaluable')
                if rdo_table_not_evaluable:
                     RDO='DIF_CONJ'
                     num_DIF_CONJ+=1
                else:
                     rdo_table_fail = encontrar_form_table(form, etiqueta,
lema, root, cod_lema, cursor, 'table_fail_real')
                     if rdo_table_fail:
                          RDO='FAIL'
                          num_FAIL+=1
                     else:
                          rdo_table_correct = encontrar_form_table(form,
etiqueta_elixir, lema, root, cod_lema, cursor, 'table_correct')
                          if rdo_table_correct:
                               RDO='CORRECT'
                               num_CORRECT+=1
                          else:
                               evaluado=form_to_eval(form, etiqueta,
cod_lema, lema, root, cursor)
                               if evaluado=='NO':
                                    RDO='NO_EVAL'
                                    num_NO_EVAL+=1
                               else:
                                    RDO='NO_FOUND'
                                    num_NO_FOUND+=1


                add_eval_in_table(form, form, etiqueta, cod_lema, lema, root,
cursor, 'table_freq', rdo=RDO)

     print('_________________EVALUATION VERBOS MÁS FRECUENTES_________________')
     print('NÚMERO DE NO_FOUND:      ' + str(num_NO_FOUND))
     print('NÚMERO DE NO_EVAL:    ' + str(num_NO_EVAL))
     print('NÚMERO DE CORRRECT:      ' + str(num_CORRECT))
     print('NÚMERO DE FAIL:      ' + str(num_FAIL))
     print('NÚMERO DE DIF_CONJ:      ' + str(num_DIF_CONJ))
     print('____________________________________________________________________')


     connection_jabalin.commit()
```

```
    connection_jabalin.close()
```

utilitues_data.py

```
def preprocess_lexicons():

    eq_codes={
    '00L0001':'Iau',
    '00L0000':'Iai',
    '00L0002':'Iaa',
    '00L0101':'Iuu',
    '00L0202':'Iia',
    '00L0200':'Iii',
    '20H0010':'II',
    '30H0010':'III',
    '03H0010':'IV',
    '20H1002':'V',
    '30H1002':'VI',
    '01L0000':'VII',
    '02L0000':'VIII',
    '10L0000':'IX',
    '04H0000':'X',
    '15H0000':'XI',
    '56H0000':'XII',
    '07H0000':'XIII',
    '18H0000':'XIV',
    '48H0000':'XV',
    '00H0010':'QI',
    '00H1002':'QII',
    '08H0000':'QIII',
    '10H0000':'QIV',
    }

    ## lexicon lemmas
    with open('GenerationVerbs\lexicon_lemas_jabalin.txt',encoding='utf8')
as file,\

open('DataExtraction\lexicon_lemas_procesado.txt','w',encoding='utf8') as
outfile:
        for line in file:
            try: l,r,c=line.strip().split()
            except: print(line)
            c=eq_codes[c]
```

```
            print(l,r,c, sep='\t' ,file=outfile)


    ## lexicon verbs
    with open('ResultadosFile\lexiconVerbs_jabalin.txt',encoding='utf8') as
file,\

open('DataExtraction\lexiconVerbsPerf_procesado.txt','w',encoding='utf8') as
outfileP,\

open('DataExtraction\lexiconVerbsImperf_procesado.txt','w',encoding='utf8')
as outfileI:
        for line in file:
            try: f,t,l,r,c=line.strip().split()
            except: print(line)
            c=eq_codes[c]
            if t=='VPAN3SM':
                 print(f,l,r,c, sep='\t' ,file=outfileP)
            if t=='VIAN3SM':
                print(f,l,r,c, sep='\t' ,file=outfileI)

    return()




def saca_root_patterns(length_root):
    'creates a dic with the data in lexicon of lemmas'
    ROOTS={} # dic ROOTS -> v=root; k=[code, code, ...]
    with open('DataExtraction\lexicon_lemas_procesado.txt',encoding='utf8')
as f:
        for line in f:
            try: l,r,p=line.strip().split()
            except: print(line)
            if length_root==len(r):
                if r in ROOTS: ROOTS[r].append(p)
                else: ROOTS[r]=[p]
    return ROOTS



def freq_dic(dic,total='None'):
    '''takes a dic containing {item : abs_freq}
    creates a dic -> {item : (abs_freq, %_freq)}'''
    FREQ={}
```

```
    for k,v in sorted(dic.items(), key=lambda x:x[1], reverse=True):
        if total=='None': FREQ[k]=v
        else: FREQ[k]=(v,float('%.1f' % round(v*100/total,1)))
    return FREQ



def cmp_to_key(mycmp):
    'Convert a cmp= function into a key= function'
    class K(object):
        def __init__(self, obj, *args):
            self.obj = obj
        def __lt__(self, other):
            return mycmp(self.obj, other.obj) < 0
        def __gt__(self, other):
            return mycmp(self.obj, other.obj) > 0
        def __eq__(self, other):
            return mycmp(self.obj, other.obj) == 0
        def __le__(self, other):
            return mycmp(self.obj, other.obj) <= 0
        def __ge__(self, other):
            return mycmp(self.obj, other.obj) >= 0
        def __ne__(self, other):
            return mycmp(self.obj, other.obj) != 0
    return K


def numeric_compare(x, y):
    'sort list by the order defined in the list below'

orden=['Iau','Iai','Iaa','Iuu','Iia','Iii','II','III','IV','V','VI','VII',
       'VIII','IX','X','XI','XII','XIII','XIV','XV','QI','QII','QIII','QIV']
    return orden.index(x) - orden.index(y)


def printDic_ordenado(Dic):
    '''prints ditionary sorted by pattern order'''
    ##  {IX : {'XIII': (abs, 0.0), ....}, ...}
    orden_patterns_x=sorted(Dic, key=cmp_to_key(numeric_compare))
    for pat_x in orden_patterns_x:
        dict_valores=Dic[pat_x]
        orden_patterns_y=sorted(Dic[pat_x], key=cmp_to_key(numeric_compare))
        for pat_y in orden_patterns_y:
            value_abs=dict_valores[pat_y][0]
            value_frq=dict_valores[pat_y][1]
            print(pat_x, pat_y, value_abs, value_frq, sep='\t')
```

```
    return



def saca_perfective_forms(VarForm):
    '''extracts perfective of imperfective forms
    from Jabalín lexicon of inflected verbal forms'''
    VERBS={} # {pat: [form, form, ...], ...}
    if VarForm=='1':
input_file='DataExtraction\lexiconVerbsPerf_procesado.txt'
    elif VarForm=='2':
input_file='DataExtraction\lexiconVerbsImperf_procesado.txt'
    with open(input_file,encoding='utf8') as file:
        for line in file:
            try: f,l,r,p=line.strip().split()
            except: print(line)
            VERBS.setdefault(p,[f]).append(f)
    return VERBS
```

utilities.php

```
<?php

/**************************************************************************
*****************/
/*                                    FUNCIONES DE TRADUCCIÓN
*/
/**************************************************************************
*****************/
// función que traduce las etiquetas a texto
function traduceEtiqueta($etiqueta){
       $trad_Pos1 = array('P'=> 'perfective', 'I'=>'imperfective');
       $trad_Pos2 = array('A'=> 'active', 'P'=>'passive');
       $trad_Pos3 = array('N'=> 'indicative', 'S'=>'subjuntive',
'Y'=>'jussive', 'M'=>'imperative');
       $trad_Pos4 = array('1'=> 'first person', '2'=>'second person',
'3'=>'third person');
       $trad_Pos5 = array('S'=> 'singular', 'D'=>'dual', 'P'=>'plural');
       $trad_Pos6 = array('M'=> 'masculine', 'F'=>'feminine', 'N'=>'');

       $prim = $trad_Pos1[$etiqueta[1]].' ';
       $segun =  $trad_Pos2[$etiqueta[2]].' ';
       $ter =  $trad_Pos3[$etiqueta[3]].' ';
       $cuart = $trad_Pos4[$etiqueta[4]].' ';
```

```
        $quint = $trad_Pos5[$etiqueta[5]].' ';
        $sex = $trad_Pos6[$etiqueta[6]];
        if ($ter == 'imperative '){
                $solucion= ($ter.$cuart.$quint.$sex);
        }
        else{$solucion= ($prim.$segun.$ter.$cuart.$quint.$sex);}
        return $solucion;
}


// función traduce código
function traduceCodigo($codigo){
$traducionCod = array("00L0001"=>'Iau 00','فعَل يفعُلL0000"=>'Iai  فعَل
00','يفعِلL0002"=>'Iaa 00','فعَل يفعَلL0101"=>'Iuu 00','فعُل يفعُلL0202"=>'Iia
00','فعِل يفعَلL0200"=>'Iii 20','فعِل يفعِلH0010"=>'II 30" ,'فعَلH0010"=>'III
03','فاعلH0010"=>'IV 20','أفعلH1002"=>'V 30','تفعّلH1002"=>'VI
01','تفاعلL0000"=>'VII 02','انفعلL0000"=>'VIII 10','افتعلL0000"=>'IX
04','افعلّH0000"=>'X 15','استفعلH0000"=>'XI 56','افعالّH0000"=>'XII
07','افعوعلH0000"=>'XIII 18','افعوّلH0000"=>'XIV 48','افعنللH0000"=>'XV
00','افعنلىH0010"=>'QI 00','فعللH1002"=>'QII 08','تفعللH0000"=>'QIII
10','افعنللH0000"=>'QIV افعللّ');
return $traducionCod[$codigo];
}


/*************************************************************************
*****************/
/*                                        FUNCIONES DE COMPARACIÓN
*/
/* para ordenar los códigos de lema y sus patterns correspondientes
*/
/*************************************************************************
*****************/

// función que compara dos codes y devuelve menos que cero si el segundo va
antes que el primero
// y mayor que cero en el caso contrario. Si son el mismo devuelve cero
function cmp($a, $b){
        // $orden =
array('Iau'=>0,'Iai'=>1,'Iaa'=>2,'Iuu'=>3,'Iia'=>4,'Iii'=>5,'II'=>6,'III'=>7
,'IV'=>8,'V'=>9,'VI'=>10,'VII'=>11,
        //
'VIII'=>12,'IX'=>13,'X'=>14,'XI'=>15,'XII'=>16,'XIII'=>17,'XIV'=>18,'XV'=>19
,'QI'=>20,'QII'=>21,'QIII'=>22,'QIV'=>23);
```

```
        $orden =
array("00L0001"=>0,"00L0000"=>1,"00L0002"=>2,"00L0101"=>3,"00L0202"=>4,"00L0
200"=>5,"20H0010"=>6,
"30H0010"=>7,"03H0010"=>8,"20H1002"=>9,"30H1002"=>10,"01L0000"=>11,"02L0000"
=>12,"10L0000"=>13,"04H0000"=>14,"15H0000"=>15,"56H0000"=>16,"07H0000"=>17,"
18H0000"=>18,"48H0000"=>19,"00H0010"=>20,"00H1002"=>21,"08H0000"=>22,"10H000
0"=>23);

        return ($orden[$a]-$orden[$b]);
}
// Función para comparar dos codes en flexionar lema
function cmp_root($a, $b){
        // llegan los datos lema pattern y sólo nos interesa la primera
parte...hacemos un split
        $at = explode(' ',$a, 2);
        $ap = $at[1];
        $bt = explode(' ',$b, 2);
        $bp = $bt[1];

        return (cmp($ap, $bp));
}

// Función para compara dos codes en analizar forma
function cmp_form($a, $b){

        $ap = $a['cod_lema'];
        $bp = $b['cod_lema'];

        return (cmp($ap, $bp));
}


/**************************************************************************
*****************/
/*                                        FUNCIONES GENERALES
*/
/**************************************************************************
*****************/

// función que quita las vocales a una forma pasada como parámetro
function quitaVocales($form, $tipo='forma'){
        if ($tipo=='forma'){
                $vocales=array('ó', 'ï','ï','ị̈','ị̈' ,'ï' ,'ï' ,'ï''');
        }
        else{ // de momento no le quitamos la sabda a la raíz
                $vocales=array('ó', 'ï','ị̈','ị̈' ,'ï' ,'ï' ,'ï''');
```

```
	}
	foreach($vocales as &$vocal){
		$form=str_replace($vocal, "", $form);
	}
	return "$form";
}

// reemplaza hamza por otros para la raiz
function replaceHamza($word){
	$lets=array('إ','ؤ','ئ','أ');
	foreach($lets as &$l){
		$word=str_replace($l, "ء", $word);
	}
	return $word;
}


?>
```

funciones.php

```
<?php

/*************************************************************************
*****************/
/*                          FUNCIONES PRINCIPALES PARA IMPRIMIR
*/
/*************************************************************************
*****************/
function imprimeCabecera(){
	/*print'<!DOCTYPE html
	PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
	 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">';*/
	print'<html xmlns="http://www.w3.org/1999/xhtml">

	<head>
	<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
	<title>Analizador del &Aacuterabe Online JABAL&IacuteN</title>
	<link type="image/x-icon" rel="shortcut icon"
href="imagenes/favicon.ico" />
	<link rel="stylesheet" type="text/css" href="jabalin.css" />
	<style type="text/css">
	p.big {line-height:200%;}
	</style>

	</head>


	<body>
```

```
                <h1>
                        <a>JABAL&IacuteN
                        </a>
                        Online Interface of the Arabic Analyzer
                </h1>';
}

function imprimePie(){
        print('</br></br>');
        print '&copy 2012 LLI-UAM. GNU General Public License <a
href="http://www.gnu.org/licenses/">GNU GPL 3</a>.<p class="big"> Jabalín is
an <a href="http://www.lllf.uam.es/ESP/Recursos.html">open-source online</a>
project developed by Alicia González Martínez, computational linguist, and
Susana López Hervás, computer scientist, and directed by Prof. Antonio
Moreno Sandoval, principal investigator of the LLI-UAM, The Laboratorio de
Lingüística Informática, Universidad Autónoma de Madrid.</p>';
        print '</html>';
}

// Obtiene la forma del formulario
function obtieneLaNuevaForma(){
        if (isset ($_GET['formas'])){
                $form = $_GET['formas'];
        }
        else{
                $form = 'تـتمـنين';
        }
        return (trim($form));

}

// Obtiene el root del formulario
function obtieneElNuevoRoot(){
        if (isset ($_GET['raiz'])){
                $root = $_GET['raiz'];
                        }
        else{
                $root = 'قـبل';
        }
        return trim($root);

}

// Obtiene el lema del formulario
function obtieneElNuevolema(){
        if (isset ($_GET['lema'])){
                $lema = $_GET['lema'];
                        }
        else{
                $lema = 'اهتمّ';
        }
        return trim($lema);

}
// calcula el nuevo offset cuando se  ha presionado los botones de avanzar o
retroceder
function calculaElNuevoOffset($numFormas, $tipopag=''){
        if (isset ($_GET['offset'])){
                $offset = trim($_GET['offset']);
        }
```

```
        else{
                $offset = 0;
        }
        $offset_back = $offset - 25;
        $offset_next = $offset + 25;

        // Evitamos que el offset sea negativo
        if ($offset_back < 0){
                $offset_back = 0;
        }
        // Evitamos que se flipe en las consultas
        if ($offset_next > ($numFormas)){
                $offset_next = $numFormas;
                $offset_back = $numFormas-25;
        }
        if ($offset > ($numFormas)){
                $offset = $numFormas;
        }
        if ($tipopag=="BBDD"){
                $off=$offset-1;
        }
        if($tipopag!="BBDD"){$off=$offset;}
        //return (array("back"=>$offset_back, "next"=>$offset_next,
"actual"=>($offset-1)));
        return (array("back"=>$offset_back, "next"=>$offset_next,
"actual"=>($off)));
}

// imprime el formulario del offset para mostrar base de datos.
function imprimirBotonOffset($offsets,$numFormas){
        print('<table border="0">');
        print('<tbody>');
        print('<tr align="left">');
        // print('<td class=BdD align="left">');
        // print ('<b>'. $numFormas.'</b> Total forms');
        // print('</td>');
        print('<td class=Offset align="left">');
        echo '<form name="input" action="mostrarBdD.php" method="get">';
        $offset_a=$offsets['actual']+1;
        if ($offset_a<=0){$offset_a=1;}

        echo 'id: <input type="text" size="7" maxlength="9" name="offset"
value="' . $offset_a  . '" /> of <b>'.$numFormas.'</b> total forms ';
        echo '<input type="submit" value="Update" />';
        echo '</form>';
}

// return (array("numFormas"=>$numForms, "numLemas"=>$numLemas,
"numRaices"=>$numRoot,
"numFomasPorlema"=>109,"forms_evaluables"=>$forms_evaluables,"correctos"=>$c
orrectos, "fallos"=>$fallos, "no_eval"=>$no_eval,
"%Aciertos"=>$PorcAciertos,"%Fallos"=>$PorcFallos ));
function imprimir_info_general_BdD($datos){
        print('<table border="0">');
        print('<tbody>');
        print('<tr align="left">');
        print('<td class=BdD align="left">');
        print ('<b></b>');
        print ('<b>109</b> forms/lemma');
        echo '<br />';
        while ($row = $datos->fetchArray()) {
```

```
                // echo $row[1]."->".$row[2];
                // echo '<br />';
                // val_dump($row);
                if ($row[1]=='numLemas'){
                        print ('<b>'. $row[2].'</b> lemmas');
                        echo '<br />';
                }
                if ($row[1]=='numRaices'){
                        print ('<b>'.$row[2].'</b> roots');
                        // echo '<br />';
                }
        }
        $datos->reset();
        print('</td>');
}

function imprimir_info_Eval_BdD($datos){
        print('<td class=BdD>');
        while ($row = $datos->fetchArray()) {
                // echo $row[1]."->".$row[2];
                // echo '<br />';
                // val_dump($row);
                if ($row[1]=='forms_evaluables'){
                        print ('<b>'. $row[2].'</b> evaluable forms');
                        echo '<br />';
                }
                if($row[1]=='correctos'){
                        print ('<b>'. $row[2].'</b> correct forms');
                        echo '<br />';
                }
                if($row[1]=='%Aciertos'){
                        print ('<b>'. $row[2].'</b> %correct forms');
                        // echo '<br />';
                }
        }
        $datos->reset();
        print('</td>');
        print('<td></td>');
}

// imprime el formulario del offset para mostrar base de datos.
function imprimirBotonChoise($nombre, $accion){
        echo '<form name="input" action='.$accion.'>';
        echo  '<input type="submit" value='.$nombre.' />';
        echo '</form>';
}

function imprimeMenu($opcion){
        $servidor=($_SERVER['SERVER_NAME']);
        echo '<div class="menu">';

        if ($opcion=='Home'){echo '<span>Home</span>';}
        else{ echo '<a
href="http://'.$servidor.'/jabalin/index.php">Home</a>';}
        if ($opcion=='Statistics'){echo '<span>Quantitative Data</span>';}
        else{echo '<a
href="http://'.$servidor.'/jabalin/imagenes/estadistica.pdf">Quantitative
Data</a>';}
        if ($opcion=='See'){echo '<span>Explore Database</span>';}
        else{echo '<a
href="http://'.$servidor.'/jabalin/mostrarBdD.php">Explore Database</a>';}
```

```
        if ($opcion=='Inflect'){echo '<span>Inflect verb</span>';}
        else{echo '<a
href="http://'.$servidor.'/jabalin/flexionarLema.php">Inflect verb</a>';}
        if ($opcion=='Derive'){echo '<span>Derive root</span>';}
        else{echo '<a
href="http://'.$servidor.'/jabalin/derivarRaiz.php">Derive root</a>';}
        if ($opcion=='Analyze'){echo '<span>Analyze form</span>';}
        else{echo '<a
href="http://'.$servidor.'/jabalin/analizarForma.php">Analyze form</a>';}
        echo '</div>';
}

// función que imprime el formulario de consulta
function imprimirFormaConsulta($nombre, $accion, $VariableBusqueda,
$nombreBoton){
print ('<form>');
        print('<table border="0">');
        print('<tbody>');
        print('<tr align="left">');
        print('<td align="left">');
        echo '<form name="input" action='. $accion .' method="get">';
        echo '<input type="text" size="10" name='. $VariableBusqueda .'
value="' . $nombre  . '" style="font-family: Traditional Arabic ; font-size:
30px;" />';
        // echo $mensaje. ': <input type="text" name='. $VariableBusqueda .'
value="' . $nombre  . '" />';
        echo ' <input type="submit" value='. $nombreBoton .' " style="font-
family: Traditional Arabic ; font-size: 20px;"/>';
        echo '</form>';
        print('</td>');
}

function imprimirBotonesAtrasSiguiente_lema($offset_back, $offset_next,
$accion, $lema,$root, $cod_lema, $infoPos=''){
        $servidor=($_SERVER['SERVER_NAME']);
        $atras="href=\"http://".$servidor."/jabalin/".$accion."?offset=".$off
set_back ."&lema=". $lema ."&raiz=".$root ."&cod_lema=". $cod_lema ."\"";
        $siguiente="href=\"http://".$servidor."/jabalin/".$accion."?offset=".
$offset_next ."&lema=". $lema ."&raiz=".$root ."&cod_lema=". $cod_lema
."\"";
        // imprimimos los botones
        print('<td align="right">');
        print '<h4>';
        print '<a '.$atras. '> ◄ </a>';
        print 'back | next';
        print '<a '.$siguiente. '> ► </a>';
        print '</h4>';
        print('</td>');
        print($infoPos);
        print('</tr>');
        print('</tbody>');
        print('</table>');
        print ('</form>');
}

// función que imprime los botones para retroceder o avanzar el offset
function imprimirBotonesAtrasSiguiente($offset_back, $offset_next, $accion,
$variable='',$nombreVar='', $infoPos=''){
        $servidor=($_SERVER['SERVER_NAME']);
        $atras="href=\"http://".$servidor."/jabalin/".$accion."?offset=".$off
set_back ."&".$nombreVar."=". $variable ."\"";
```

```
        $siguiente="href=\"http://".$servidor."/jabalin/".$accion."?offset=".
$offset_next ."&".$nombreVar."=". $variable ."\"";
        // imprimimos los botones
        print('<td align="right">');
        print '<h4>';
        print '<a '.$atras. '> ◄ </a>';
        print 'back | next';
        print '<a '.$siguiente. '> ► </a>';
        print '</h4>';
        print('</td>');
        print($infoPos);
        print('</tr>');
        print('</tbody>');
        print('</table>');
        print ('</form>');
}


// función que imprime los botones para retroceder o avanzar el offset
function imprimirBotonesAtrasSiguiente_Abajo_lema($offset_back,
$offset_next, $accion, $lema,$root, $cod_lema){
        $servidor=($_SERVER['SERVER_NAME']);
        $atras="href=\"http://".$servidor."/jabalin/".$accion."?offset=".$off
set_back ."&lema=". $lema ."&raiz=".$root ."&cod_lema=". $cod_lema ."\"";
        $siguiente="href=\"http://".$servidor."/jabalin/".$accion."?offset=".
$offset_next ."&lema=". $lema ."&raiz=".$root ."&cod_lema=". $cod_lema
."\"";
        //imprimimos los botones
        print '<h4>';
        print '<a '.$atras. '> ◄ </a>';
        print 'back | next';
        print '<a '.$siguiente. '> ► </a>';
        print '</h4>';
}

// función que imprime los botones para retroceder o avanzar el offset
function imprimirBotonesAtrasSiguiente_Abajo($offset_back, $offset_next,
$accion, $variable='',$nombreVar=''){
        $servidor=($_SERVER['SERVER_NAME']);
        $atras="href=\"http://".$servidor."/jabalin/".$accion."?offset=".$off
set_back ."&".$nombreVar."=". $variable ."\"";
        $siguiente="href=\"http://".$servidor."/jabalin/".$accion."?offset=".
$offset_next ."&".$nombreVar."=". $variable ."\"";
        //imprimimos los botones
        print '<h4>';
        print '<a '.$atras. '> ◄ </a>';
        print 'back | next';
        print '<a '.$siguiente. '> ► </a>';
        print '</h4>';
}

/*************************************************************************
*****************/
/*                          FUNCIONES SECUNDARIAS PARA IMPRIMIR
*/
/*************************************************************************
*****************/
// Función que imprime dentro de la tabla de resultados el contenido del
array $file, teniendo en cuenta
// los elementos contenidos en el array $elementos. Si uno de los elementos
es etiqueta y se
```

```
// la variable $traducirCod es "si" la traduce a texto
function imprimir_row($fila, $elementos, $traducirCod="SI"){
        print "\t\t\t<tr>\n";
        $clases = array("id"=>'id',"form_without_vowels"=>'orth',
        "form"=>'orth', "lema"=>'leme', "root"=>'root',
        "pattern"=>'morphs',"cod_lema"=>'code',"etiqueta"=>'xtag',"eval"=>'cl
ass');
        foreach($elementos as &$element){
                if ($element ==''){
                        print "<td class='xtag'> </td>";
                }
                else{
                        $resul_element = $fila[$element];
                        if (($traducirCod=="SI")&&($element == "cod_lema")){
                                $element = "pattern";
                                $resul_element = traduceCodigo($resul_element);
                        }
                        if($resul_element=='SI'){
                                $resul_element='YES';
                        }
                        else if($resul_element==''){
                                $resul_element='NO';
                        }

                        print "<td class=".$clases[$element]."> " .
$resul_element ." </td>";
                        if ($element == "etiqueta"){
                                print "<td class='ttag'> " .
traduceEtiqueta($resul_element) ." </td>";
                        }
                //echo "$element: " . $resul_element . "<br />";
                }
        }
        print "\t\t\t</tr>";
}

//************************************ DERIVE ROOT
*********************************//

// Función que imprime el resultado de una consulta por raiz
function imprimir_results_raiz($resultados, $elementos, $offset, $limit){
        $i=0;
        $results=array();
        while ($row = $resultados->fetchArray()) {
                $i++;
                $results[$i]=$row['concatenado'];
        }
        $results=array_unique($results);
        // var_dump($results);

        // ordenamos por pattern
        usort($results, "cmp_root");

        foreach($results as $r){
                $rest_split = explode(' ',$r, 2);
                imprimir_row(array('lema'=>$rest_split[0],
'cod_lema'=>$rest_split[1]), $elementos);
        }
        print "\t\t</tbody>";
        print "\t</table>";
        print "</li>";
```

```
        print "</ul>";
}

//************************************ INFLECT VERB
*********************************//

// Función que imprime la primera fila con el lema, raiz y patter de la
consulta por lema
function imprimir_lema($forma,$resultados){
        $servidor=($_SERVER['SERVER_NAME']);
        print "<ul class='listenTable'>";
        print "<li class='listenTable'>";
        print "\t<table class='lexeme' cellspacing='0'>";
        $row = $resultados->fetchArray();
        if($row){
                $lema = $row["lema"];
                $root = $row["root"];
                $pattern = traduceCodigo($row["cod_lema"]);
                $resultados->reset();

                print "<td title=\"verb\" class=consult_lema>" . $lema."
</td>";
                print "<td title=\"root\" class=consult_raiz>" .
$root."</td>";
                print "<td title=\"pattern\" class=consult_pattern>" .
$pattern. "</td>";
                if (isset($_GET['raiz'])){
                        print '<td class=consult_link><a
href="http://'.$servidor.'/jabalin/flexionarLema.php?lema='.$_GET['lema'].'"
>return to verbs</a></td>';
                }
                else{
                        print"<td class=consult> </td>";
                }
        }
        else{
                print"<td class=consult> no more forms </td><td
class=consult_raiz></td><td class=consult_pattern></td>";
                if (isset($_GET['raiz'])){
                        print '<td class=consult_link><a
href="http://'.$servidor.'/jabalin/flexionarLema.php?lema='.$_GET['lema'].'"
>return to verbs</a></td>';
                }
                else{
                        print"<td class=consult> </td>";
                }
        }
        print "\t\t<tbody>";

}


// Función que imprime el resultado de una consulta por lema
function imprimir_cada_lema($resultados){
        $servidor=($_SERVER['SERVER_NAME']);
        print "<ul class='listenTable'>";
        print "<li class='listenTable'>";

        print "\t<table class='lexeme' cellspacing='0'>";

        $lema = $resultados["lema"];
```

```
        $root = $resultados["root"];
        $cod_lema = $resultados["cod_lema"];
        $pattern = traduceCodigo($cod_lema);

        print "<td title=\"verb\" class=consult_lema>" . $lema." </td>";
        print "<td title=\"root\" class=consult_raiz>" . $root."</td>";
        print "<td title=\"pattern\" class=consult_pattern>" . $pattern.
"</td>";
        print '<td class=consult_link><a
href="http://'.$servidor.'/jabalin/flexionarLema.php?lema='.$lema.'&raiz='.$
root.'&cod_lema='.$cod_lema.'">see inflectional paradigm</a></td>';
        print "\t\t</table>";
        print "\t\t</li>";
        print "\t\t</ul>";
        print '<p>';
}

// Función que imprime el resultado de una consulta por lema
function imprimir_Vbs_por_lema($resultados, $lema, $offsets, $limit){
        $i=0;
        $results=array();
        while ($row = $resultados->fetchArray()) {
                $i++;
                $results[$i]=$row['concatenado'];
        }
        // si sólo hay un verbo por lema no hay que elegir nada
        $results=array_unique($results);
        // var_dump($results);
        // si hay más de uno
        if (count($results)==1){
                // var_dump($results);
                imprimirBotonesAtrasSiguiente($offsets["back"],
$offsets["next"], "flexionarLema.php", $lema, 'lema', $infoPos='<td
align="right"> <b>'.$offsets['next'].'</b> of <b>109</b> total
forms/verb</td>');
                mostrar_por_lema($lema, $offsets['actual'], $limit);
                imprimirBotonesAtrasSiguiente_Abajo($offsets["back"],
$offsets["next"], "flexionarLema.php", $lema, 'lema');
        }
        else{
                print('</tr>');
                print('</tbody>');
                print('</table>');
                print ('</form>');
                foreach($results as $r){
                        // print($r);
                        $rest_split = explode(' ',$r, 3);
                        $r=array('lema'=>$rest_split[0],
'cod_lema'=>$rest_split[1], 'root'=>$rest_split[2]);
                        // var_dump($r);
                        imprimir_cada_lema($r);
                }
        }
        //var_dump($results);
}


//********************************** EXPLORE DATABASE
*********************************//

// Función que imprime la cabecera de una consulta
```

```
function imprimir_cabecera_tabla($elementos, $traducirCod="NO"){
	foreach($elementos as &$element){
		if (($traducirCod=="SI")&&($element == "cod_lema")){
			$element = "pattern";
		}
		if ($element == "form"){
			$element = "vocalized_form";
		}
		if ($element == "form_without_vowels"){
			$element = "form";
		}
		if ($element == "cod_lema"){
			$element = "code";
		}
		if ($element == "lema"){
			$element = "lemma";
		}
		if ($element == "eval"){
			print "<td class=cab> translated_tag </td>";
		}
		if ($element == 'etiqueta'){$element='tag';}
		// print "<td class=".$clases[$element]."> " . $element ."
</td>";
		print "<td class=cab> " . $element ." </td>";
	}
}

// Función que imprime el resultado de una consulta geneal a la BdD
function imprimir_results_BdD($resultados, $elementos){
	print "<ul class='listenTable'>";
	print "<li class='listenTable'>";
	print "\t<table class='lexeme' cellspacing='0'>";
	imprimir_cabecera_tabla($elementos, $traducirCod="NO");
	print "\t\t<tbody>";
	while ($row = $resultados->fetchArray()) {
		imprimir_row($row, $elementos, $traducirCod="NO");
	}
	print "\t\t</tbody>";
	print "\t</table>";
	print "</li>";
	print "</ul>";
}

//************************************ ANALYZE FORM
**********************************//

// función que devuelve "SI" si $forma es una forma vocalizada válida para
la forma
// semivocalizada $formaMV. Devuelve "NO" si no es válida
function esValida($formaMV, $forma){
	// $vocales=array('a','e','i','o');
	$leter= '';
	$vocales=array('ó', 'ِ','ُ','ٍ' ,'ٌ' ,'ً' ,'');
	$i=0;
	$simbolo='ó';
	// $simbolo='w';
	// كتبَ
	$let_ant='';

	while ($i<strlen($formaMV)){
		$letraBuscar=($formaMV[$i].$formaMV[$i+1]);
```

```
                if((in_array($letraBuscar, $vocales))&&($let_ant==$simbolo)){
                        return "NO";
                }
                $rest_split = explode($letraBuscar,$forma, 2);

                // si no hace split es que no ha encontrado la letra
                if ($rest_split[0]==$forma){
                        return "NO";
                }
                $forma = $rest_split[1];
                if($forma!=''){ $let_ant=($forma[0].$forma[1]);}
                // echo $forma. '<br />';
                $i++;
                $i++;
        }
        return "SI";

}

// función que devuelve una array con las formas vocalizadas que sean
compatibles con la forma
// semivocalizada $fromMv conteidas en un array de formas posibles pasado
como parámetro $formas
function formas_compatibles($formaMV, $result_query){
        $lista_resultados='';
        while ($row = $result_query->fetchArray()){
                if (esValida($formaMV, $row['form']) == "SI"){
                        $lista_resultados[] = $row;
                }
        }
        return $lista_resultados;
}

// Función que imprime la forma que se ha dado a analizar en la tabla
function imprimir_form($forma, $class_f,$title){
        print "<ul class='listenTable'>";
        print "<li class='listenTable'>";
        print "\t<table class='lexeme' cellspacing='0'>";
                print "<td title=\"".$title."\" class=".$class_f.">" .
$forma." </td>";
                print "<td class=consult> </td>";
                print "<td class=consult> </td>";
                print "<td class=consult>  </td>";
                print "<td class=consult> </td>";
                if ($title=='form'){
                        print "<td class=consult> </td>";}

        print "\t\t<tbody>";

}

// Función que imprime el resultado de una consulta para analizar forma.
Ordena los resutados por pattern
function imprimir_results($resultados, $elementos, $offset, $limit){
        $i=0;
        while ($row = $resultados->fetchArray()) {
                $results[$i] = $row;
                $i++;
        }
        if ($i!=0){
```

```
                usort($results,"cmp_form"); // ordena los resultados por
pattern
                foreach($results as $r){
                        imprimir_row($r, $elementos);
                }
        }
        print "\t\t</tbody>";
        print "\t</table>";
        print "</li>";
        print "</ul>";
}

// Función que imprime el resultado de una consulta de una forma vocalizada
para analizar forma
function imprimir_results_vocalizada($results, $elementos, $offset, $limit){
        usort($results, "cmp_form");
        $i=0;
        if($results){
                foreach ($results as &$r){
                        $i++;
                        if (($i>$offset-1)&&($i<($offset+$limit))){
                                // var_dump($r);
                                imprimir_row($r, $elementos);
                        }
                }
        }
        print "\t\t</tbody>";
        print "\t</table>";
        print "</li>";
        print "</ul>";
}


// Función que imprime los resultados de una consulta por forma
function imprimir_results_BdD_forma($results, $SemiVocalizada="NO", $offset,
$limit){
        $elementos = array("form", "lema", "root", "cod_lema", "etiqueta");
        $elementos_cabecera = array("form", "lema", "root", "cod_lema",
"etiqueta","");
        imprimir_cabecera_tabla($elementos_cabecera, $traducirCod="SI");
        // ordenamos por pattern
        if ($SemiVocalizada=="SI"){
                imprimir_results_vocalizada($results, $elementos, $offset,
$limit);
        }
        else{
                imprimir_results($results, $elementos, $offset, $limit);
        }
}

// <tr align="left">
// <td align="left">
// <input type="submit" value="Resolve" name="submit">
// </td>
// <td align="center">
// <input type="button" onclick="elixirClear('text')" value="Clear"
name="clear">
// </td>

// imprimirBdD();
```

```
// imprimirBdD();
?>
```

mostrarBdD.php

```
<?php
	include("funciones.php");
	include("bbdd.inc.php");

	$bd = new SQLite3('lexiconVerbsdata_Jabalin.sqlite');
	guardar_info_general_Eval_BdD($bd);
	$infoBdD = recuperar_Estadisticas($bd);

	$numFormas = numTotalFormas();
	// return (array("numFormas"=>$numForms, "numLemas"=>$numLemas,
"numRaices"=>$numRoot,
"numFomasPorlema"=>109,"forms_evaluables"=>$forms_evaluables,"correctos"=>$c
orrectos, "fallos"=>$fallos, "no_eval"=>$no_eval,
"%Aciertos"=>$PorcAciertos,"%Fallos"=>$PorcFallos ));

	imprimeCabecera();
	imprimeMenu('See');

	$offsets = calculaElNuevoOffset($numFormas, "BBDD");
	imprimirBotonOffset($offsets,$numFormas);
	imprimirBotonesAtrasSiguiente($offsets["back"], $offsets["next"],
"mostrarBdD.php");
	mostrar_BdD($offsets["actual"], $lim = 25);
	// imprimirTablasEstadistic();
	print('</br>');

	imprimir_info_general_BdD($infoBdD);
	imprimir_info_Eval_BdD($infoBdD);
	imprimirBotonesAtrasSiguiente($offsets["back"], $offsets["next"],
"mostrarBdD.php");
	$bd->close();

	imprimePie();
	?>
```

index.php

```
<?php
	include("funciones.php");
	include("bbdd.inc.php");
	$servidor=($_SERVER['SERVER_NAME']);

	imprimeCabecera();
	imprimeMenu('Home');

	print '<br/><br/>';
	print '<p class="big"><bb>Jabal&iacuten</bb> is an application for
analyzing and generating verbs in Modern Standard Arabic. It uses a large-
scale lexicon of inflected forms which has been generated following a root-
and-pattern approach.  The system provides three functionalities: <b>inflect
verb</b>, <b>derive root</b> and <b>analyze form</b>. Besides, the
application offers the possibility to <b>explore the database</b> containing
the lexicon of verbs and additional information. It also provides
<b>quantitative data</b> extracted from the lexicons that can be used to
perform statistical analysis on Arabic morphology.</bb></p>';
	// imprimirBotonHome("HOME", "index.php");
	print '</br>';
	print '<p>';
	print '<h2>
				<a
href="http://'.$servidor.'/jabalin/mostrarBdD.php">Explore Database</a>
			</h2>';

	print  '<p>
				This options allows you to look into the
lexicon of Jabal&iacuten.
			</p>';
	print '</br>';
	print '<p>';
	print '<h2>
				<a
href="http://'.$servidor.'/jabalin/imagenes/estadistica.pdf">Quantitative
Data</a>
			</h2>';
	print  '<p>
				This options provides quantitative data
extracted from the Jabal&iacuten lexicons, the lexicon of verbal lemmas and
the lexicon of inflected forms.
			</p>';
	print '</br>';
	print '<h2>
```

```
                        <a
href="http://'.$servidor.'/jabalin/flexionarLema.php">Inflect verb</a>
                    </h2>';
        print  '<p>
                        This functionality provides the conjugation
paradigm of a given verb lemma. If the verb has shadda, it must be written!
                    </p>';
        print '</br>';
        print '<h2>
                        <a
href="http://'.$servidor.'/jabalin/derivarRaiz.php">Derive root</a>
                    </h2>';
        print  '<p>
                        This functionality lists all the verb lemmas
generated from a given root.
                    </p>';
        print '</br>';
        print '<h2>
                        <a
href="http://'.$servidor.'/jabalin/analizarForma.php">Analyze form</a>
                    </h2>';
        print  '<p>
                        This functionality provides all the possible
analyses of a given verb form. It accepts fully vocalized, partially
vocalized or unvocalized forms.
                    </p>';
        print '</br>';
        print '</br>';

        imprimePie();
        ?>
```

analizarForma.php

```
<?php
        include("funciones.php");
        include("bbdd.inc.php");

        imprimeCabecera();
        imprimeMenu('Analyze');

        $form = obtieneLaNuevaForma();
        imprimirFormaConsulta($form, "analizarForma.php", "formas",
"Analyze_form");
```

```
        $offsets = calculaElNuevoOffset(50);

        imprimirBotonesAtrasSiguiente($offsets["back"], $offsets["next"],
"analizarForma.php", $form, 'formas');

        analizar_forma ($form, $offsets['actual'], $limit=25);

        imprimirBotonesAtrasSiguiente_Abajo($offsets["back"],
$offsets["next"], "analizarForma.php", $form, 'formas');


        imprimePie();
        ?>
```

bbdd.inc.php

```
<?php
//error_reporting(0);
include("utilities.php");



//************************************ INFORMACIÓN GENERAL
**********************************//

// función que retorna el número de filas distintas que hay en una columna
dada
function numRowsDistintas($bd, $columna){
        return ($bd->querySingle('SELECT COUNT(DISTINCT '.$columna.') FROM
lexiconVerbs'));
}

// función que devuelve el número de filas que hay en una tabla
function numRows($bd, $table){
        return ($bd->querySingle('SELECT COUNT(*) FROM '.$table));
}

function numTotalFormas(){
        $bd = new SQLite3('lexiconVerbsdata_Jabalin.sqlite');
        $numForms = numRows($bd, 'lexiconVerbs');
        $bd->close();
        return $numForms;
```

```
}


//************************************ DATOS ESTADÍSTICOS
**********************************//

// función que devuelve un array con información general de la base de datos
necesaria para crear la tabla de estadísticas
// array("numFormas"->x, "numLemas"->y, "numRaices"->z, "numFomasPorlema"-
>w)
function extraer_y_crear_tabla_info_general_BdD($bd){
	// obtenemos la información
	$numForms = numRows($bd, 'lexiconVerbs');
	// posible número de lemas puesto a mano por ser diferente el código
de lema: 15453
	//$numLemas = $bd->querySingle('SELECT COUNT(DISTINCT lema) FROM
lexiconVerbs');
	$numLemas = $bd->querySingle('SELECT COUNT(DISTINCT (lema || cod_lema
|| root)) from lexiconVerbs');
	$numRoot = numRowsDistintas($bd, 'root');
	// sacamos la información de evaluación
	$forms_evaluables= $bd->querySingle('SELECT COUNT(*) FROM
lexiconVerbs WHERE eval="SI"');
	$correctos = numRows($bd, 'table_correct');
	$fallos = numRows($bd, 'table_fail_real');
	$no_eval = numRows($bd, 'table_not_evaluable');

	$PorcAciertos = (round(($correctos*100/$forms_evaluables)*100))/100;
	$PorcFallos = (round(($fallos*100/$forms_evaluables)*100))/100;
	return (array("numFormas"=>$numForms, "numLemas"=>$numLemas,
"numRaices"=>$numRoot,
"numFomasPorlema"=>109,"forms_evaluables"=>$forms_evaluables,"correctos"=>$c
orrectos, "fallos"=>$fallos, "no_eval"=>$no_eval,
"%Aciertos"=>$PorcAciertos,"%Fallos"=>$PorcFallos ));
}

function recuperar_Estadisticas($bd){
	$r = $bd->query("SELECT * FROM table_info");
	return $r;
}

// función que guarda información en la base de datos
function inicializar_BdD($array_info, $bd){
	$elementos=array_keys($array_info);
	foreach($elementos as &$elemento){
```

```
                $bd->exec("INSERT INTO table_info (Descripcion, Valor)
VALUES(\"".$elemento."\", ".$array_info[$elemento].")");
        }
}

// función que guarda información general de estadística en la base de
datos, en caso de que no exista.
function guardar_info_general_Eval_BdD($bd){
        // Tenemos que comprobar que la tabla de información existe
        $stmt = $bd->prepare('SELECT * FROM table_info');
        if ($stmt == false) { // No existe, así que la creamos e
inicializamos
                $bd->exec('CREATE TABLE table_info(id INTEGER PRIMARY KEY
AUTOINCREMENT UNIQUE NOT NULL, Descripcion TEXT NOT NULL, Valor INTEGER)');

                // obtenemos la información e inicializamos la tabla con ella
                $info_general = extraer_y_crear_tabla_info_general_BdD($bd);
                inicializar_BdD($info_general, $bd);
        }
}


//************************ CONSULTA GENERAL DE LA BASE DE DATOS CON OFFSET Y
LÍMITE ******************//

// Función que muetra un número de columnas, "lim", de la base de datos
'lexiconVerbsdata_Jabalin.sqlite'
// desde un punto dado, "desplaz".
function mostrar_BdD($desplaz=0, $lim=10)
{
        $bd = new SQLite3('lexiconVerbsdata_Jabalin.sqlite');

        // $results = $bd->prepare('SELECT * FROM lexiconVerbs LIMIT $1
OFFSET $2 ');
        $sentencia = $bd->prepare('SELECT * FROM lexiconVerbs LIMIT :limite
OFFSET :offset ');

        $sentencia->bindValue(':limite', $lim, SQLITE3_INTEGER);
        $sentencia->bindValue(':offset', $desplaz, SQLITE3_INTEGER);
        $results = $sentencia->execute();
        $elementos =
array("id","form_without_vowels","form","lema","root","cod_lema","etiqueta",
"eval");
        imprimir_results_BdD($results, $elementos);
        $bd->close();
}
```

```
//****************************** CONSULTAS LEMA
*********************************//

// función que escribe los resultados de verbos por lema
function choice_lema($lema, $offset, $limit){
	$lema=quitaVocales($lema, $tipo='lema');
	$bd = new SQLite3('lexiconVerbsdata_Jabalin.sqlite');
	$sentencia = $bd->prepare('SELECT (lema||\' \'||cod_lema||\'
\'||root) as concatenado FROM lexiconVerbs WHERE lema= :lema' );
	$sentencia->bindValue(':lema', $lema, SQLITE3_TEXT);
	$results = $sentencia->execute();
	$row = $results->fetchArray();
	if($row){
		$results->reset();
		imprimir_Vbs_por_lema($results,$lema, $offset, $limit);
	}
	else{
		print('</tr>');
		print('</tbody>');
		print('</table>');
		print ('</form>');

		print('Remember that shadda must be written!');
	}

	$bd->close();
}


//
function choice_lema_unique($lema, $root, $cod_lema, $offset, $limit){
	$bd = new SQLite3('lexiconVerbsdata_Jabalin.sqlite');

	// ('SELECT COUNT(DISTINCT (lema || cod_lema || root)) from
lexiconVerbs');
	// quitamos las vocales al lema
	$lema=quitaVocales($lema, $tipo='lema');

	$sentencia = $bd->prepare('SELECT id, form, root, etiqueta,
cod_lema,lema FROM lexiconVerbs WHERE lema= :lema AND root= :raiz AND
cod_lema = :codigo LIMIT :limite OFFSET :offset ');

	$sentencia->bindValue(':lema', $lema, SQLITE3_TEXT);
	$sentencia->bindValue(':raiz', $root, SQLITE3_TEXT);
```

```
        $sentencia->bindValue(':codigo', $cod_lema, SQLITE3_TEXT);
        $sentencia->bindValue(':limite', $limit, SQLITE3_TEXT);
        $sentencia->bindValue(':offset', $offset, SQLITE3_TEXT);
        $results = $sentencia->execute();
        // echo "<br> Selección por lema: ", $lema. "<br />";

        // para mostrar paginado,
        $elementos_cabecera = array("form","etiqueta", "","");
        $elementos = array("form","etiqueta");
        imprimir_lema($lema,$results);
        imprimir_cabecera_tabla($elementos_cabecera, $traducirCod="SI");
        imprimir_results($results, $elementos, $offset, $limit);
        // poner el offset-25 para la siguiente vez
        $bd->close();
}


// muestra el resultado de una consulta por lema
function mostrar_por_lema($lema, $offset, $limit)
{
        $bd = new SQLite3('lexiconVerbsdata_Jabalin.sqlite');

        // ('SELECT COUNT(DISTINCT (lema || cod_lema || root)) from
lexiconVerbs');
        // quitamos las vocales al lema
        $lema=quitaVocales($lema, $tipo='lema');

        $sentencia = $bd->prepare('SELECT id, form, root, etiqueta,
cod_lema,lema FROM lexiconVerbs WHERE lema= :lema LIMIT :limite OFFSET
:offset ');

        $sentencia->bindValue(':lema', $lema, SQLITE3_TEXT);
        $sentencia->bindValue(':limite', $limit, SQLITE3_TEXT);
        $sentencia->bindValue(':offset', $offset, SQLITE3_TEXT);
        $results = $sentencia->execute();
        // echo "<br> Selección por lema: ", $lema. "<br />";

        // para mostrar paginado,
        $elementos_cabecera = array("form","etiqueta", "","");
        $elementos = array("form","etiqueta");
        imprimir_lema($lema,$results);
        imprimir_cabecera_tabla($elementos_cabecera, $traducirCod="SI");
        imprimir_results($results, $elementos, $offset, $limit);
        // poner el offset-25 para la siguiente vez
        $bd->close();
}
```

```
//****************************** CONSULTAS POR RAÍZ
*********************************//

function mostrar_por_raiz($raiz, $offset, $limit)
{
	$bd = new SQLite3('lexiconVerbsdata_Jabalin.sqlite');

	//cambiamos la Hamza
	$raiz=replaceHamza($raiz);

	$sentencia = $bd->prepare('SELECT (lema||\' \'||cod_lema) as
concatenado FROM lexiconVerbs WHERE root= :raiz' );

	$sentencia->bindValue(':raiz', $raiz, SQLITE3_TEXT);
	$results = $sentencia->execute();
	// echo "<br> Selección por raiz: ", $raiz. "<br />";
	$elementos = array("","lema","","cod_lema");
	$elementos_cabecera = array("","lema","","cod_lema","");
	imprimir_form($raiz, 'consult_raiz', 'root');
	imprimir_cabecera_tabla($elementos_cabecera, $traducirCod="SI");
	imprimir_results_raiz($results, $elementos, $offset, $limit);

	$bd->close();
}

//****************************** CONSULTAS POR FORMA
*********************************//

function consulta_forma_en_bbdd($bd, $forma_query){
	$sentencia = $bd->prepare('SELECT id, form, lema,form_without_vowels,
root, etiqueta, cod_lema FROM lexiconVerbs WHERE form_without_vowels=
:forma');
	$sentencia->bindValue(':forma', $forma_query, SQLITE3_TEXT);
	$results = $sentencia->execute();
	return $results;
}

// Función que devuelve el análisis de la forma pasada como parámetro $forma
function analizar_forma($forma, $offset, $limit){
	$bd = new SQLite3('lexiconVerbsdata_Jabalin.sqlite');
	//primero quitamos las vocales de la forma
	$formSV = quitaVocales($forma);

	// si son iguales sacamos la lista de formas posibles
```

```
        if ($formSV == $forma){
                // echo "<br> forma sin vocalizar. <br/>";
                $Vocalizada="NO";
                $results = consulta_forma_en_bbdd($bd, $formSV);
        }

        else{
                // echo "<br> forma vocalizada. <br/>";
                $Vocalizada="SI";
                // nos hacemos con todas las posibles formas buscando por
forma no vocalizada
                $rs = consulta_forma_en_bbdd($bd, $formSV);
                // buscamos las formas compatibles con la forma parcialmente
vocalizada
                $results = formas_compatibles($forma, $rs);
        }
        // echo "LA CONSULTA DE LA FORMA ". $forma." DEVUELVE:<br /><br />";
        imprimir_form($forma,'consult_form','form');
        imprimir_results_BdD_forma($results, $Vocalizada, $offset, $limit);
        $bd->close();
}

// mostrar_BdD($desplaz=872, $lim=3);
// mostrar_por_lema("ائتبّ");
// mostrar_por_raiz("ءبد");
// quitaVocales("تَأَبَّدْنَ");
// analizar_forma("أبا");
// analizar_forma("أُبًّا");
// analizar_forma("أبَا");

?>
```

derivarRaiz.php

```
<?php
        include("funciones.php");
        include("bbdd.inc.php");

        imprimeCabecera();
        imprimeMenu('Derive');

        $root = obtieneElNuevoRoot();
```

```
        imprimirFormaConsulta($root, "derivarRaiz.php", "raiz",
"Derive_root");

        $offsets = calculaElNuevoOffset(25);

        imprimirBotonesAtrasSiguiente($offsets["back"], $offsets["next"],
"derivarRaiz.php", $root, 'raiz');
        mostrar_por_raiz ($root, $offsets['actual'], $limit=25);
        imprimirBotonesAtrasSiguiente_Abajo($offsets["back"],
$offsets["next"], "derivarRaiz.php", $root, 'raiz');


        imprimePie();
        ?>
```

flexionarLema.php

```
<?php
        include("funciones.php");
        include("bbdd.inc.php");

        imprimeCabecera();
        imprimeMenu('Inflect');
        $lema = obtieneElNuevolema();
        imprimirFormaConsulta($lema, "flexionarLema.php", "lema",
"Inflect_verb");
        $offsets = calculaElNuevoOffset(109);


        if (isset
($_GET['lema'])&&isset($_GET['raiz'])&&($_GET['cod_lema'])){
                $lema=$_GET['lema'];
                $root=$_GET['raiz'];
                $cod_lema=$_GET['cod_lema'];

                imprimirBotonesAtrasSiguiente_lema($offsets["back"],
$offsets["next"], "flexionarLema.php", $lema,$root, $cod_lema, $infoPos='<td
align="right"> <b>'.$offsets['next'].'</b> of <b>109</b> total
forms/verb</td>');

                choice_lema_unique($lema, $root, $cod_lema,
$offsets['actual'], $limit=25);
```

```
                imprimirBotonesAtrasSiguiente_Abajo_lema($offsets["back"],
$offsets["next"], "flexionarLema.php",$lema,$root, $cod_lema);
        }
        else{
                // imprimirBotonesAtrasSiguiente($offsets["back"],
$offsets["next"], "flexionarLema.php", $lema, 'lema', $infoPos='<td
align="right"> <b>'.$offsets['next'].'</b> of <b>109</b> total
forms/verb</td>');

                choice_lema($lema, $offsets, $limit=25);

                // mostrar_por_lema($lema, $offsets['actual'], $limit=25);
                // imprimirBotonesAtrasSiguiente_Abajo($offsets["back"],
$offsets["next"], "flexionarLema.php", $lema, 'lema');
        }




        imprimePie();
        ?>
```

extractData_from_lexicons.py

```
#     ____________________________________________________________
#    |  ________________________________________________________  |
#    | |                                                        | |
#    | |  EXTRACTS QUANTITATIVE DATA FROM JABALIN LEXICONS      | |
#    | |________________________________________________________| |
#    |____________________________________________________________|


#    1   number of roots, verbs and mean pattern per root
#    2   number patterns per root
#    3   freq of patterns
#    4   predicted (expected) freq of pattern co-occurrences
#    5   actual (observed) freq of pattern co-ocurrences
#    6   freq of each radical from a specified list of patterns
#    7   freq of patterns from triliteral roots that meet R2=R3 (biliterals)
#    8   freq each pattern for pat/root=1
#    9   freq of patterns from roots without Form I and QI
#    10  freq of vocalism morphemes
#    11  freq of patterns according to traditional counting of prosody
```

```
import DataExtraction.utilities_data
import itertools
import re



def quantitativeData_extractor():

    DataExtraction.utilities_data.preprocess_lexicons()  ### prepares the
lexicons to extract the data

    # dict with each root and its list of codes: {root: [pat, pat, ...]}
    RootsPatsTri=DataExtraction.utilities_data.saca_root_patterns(3)   #
triliteral
    RootsPatsQua=DataExtraction.utilities_data.saca_root_patterns(4)   #
quadriliteral

    TotalRootsTri=len(RootsPatsTri)        # number triliteral roots
    TotalRootsQua=len(RootsPatsQua)        # number quadriliteral roots


    #  _________________     _________________
    # |_________________| 1 |_________________|


    Lemas_per_rootTri,Lemas_per_rootQua=[],[]   # list with the number of
lemmas for each root
    TotalVerbsTri=0; TotalVerbsQua=0            # number of verbs

    for code_lem in RootsPatsTri.values():      # triliteral
        num_verbs_each_root=len(code_lem)
        Lemas_per_rootTri.append(num_verbs_each_root)
        TotalVerbsTri=TotalVerbsTri+num_verbs_each_root

    for code_lem in RootsPatsQua.values():      # quadriliteral
        num_verbs_each_root=len(code_lem)
        Lemas_per_rootQua.append(num_verbs_each_root)
        TotalVerbsQua=TotalVerbsQua+num_verbs_each_root



    #  _________________     _________________
    # |_________________| 3 |_________________|
```

```
    NumPatTri,NumPatQua={},{}
                    # {root: [pat, pat, ...]}
    for root,codes in RootsPatsTri.items():  # triliteral
        for cod in codes:
            NumPatTri[cod]=NumPatTri.get(cod,0)+1
            ## NumPatTri = {'I':32, 'II':120, ...}

    for root,codes in RootsPatsQua.items():  # quadriliteral
        for cod in codes:
            NumPatQua[cod]=NumPatQua.get(cod,0)+1

    freqPatTri,freqPatQua={},{} # pattern - freq abs - freq per Total roots

    for k,v in
DataExtraction.utilities_data.freq_dic(NumPatTri,TotalRootsTri).items():
        pat,num,perc=k,v[0],v[1]
        freqPatTri[pat]=(num,perc)

    for k,v in
DataExtraction.utilities_data.freq_dic(NumPatQua,TotalRootsQua).items():
        pat,num,perc=k,v[0],v[1]
        freqPatQua[pat]=(num,perc)


    #  _________________     _________________
    # |_________________| 4 |_________________|


    ## EG. FREQ OF COOCURRENCY PATTERNS II AND III
    ##
    ## pattern      abs_freq(no. verbs of this pattern)      rel_freq(over
n. total roots)
    ##  II                      1811                                    56.1
    ##  III                     996                                     30.8
    ##
    ## freq relative coocurrence II & III -> (56.1 * 30.8) / 100 = 17.2788
    ## freq absolute coocurrence II & III ->  (17.2788 * 3230{i.e.Total no.
Roots}) / 100 = 558


    def PredictCoocur(DicFreq,TotalR):


DicKeys=sorted(list(DicFreq.keys()),key=DataExtraction.utilities_data.cmp_to
_key(DataExtraction.utilities_data.numeric_compare))
```

```
pairs_dic=list(itertools.combinations(sorted(DicKeys,key=DataExtraction.util
ities_data.cmp_to_key(DataExtraction.utilities_data.numeric_compare)),2))
        # in pairs_dic we include a list of all relevant combinations of
pattern pairs [('II','III'), ('II','IV'), ...]

        CoocurFreq={}
        for pat in pairs_dic:
            x,y=pat[0],pat[1]

            # we calculate the predicted freq of each pair of patterns

resul_freq=round((float(DicFreq[x][1])*float(DicFreq[y][1]))/100,2)

            # we calculate how many roots are expected to have those
patterns
            resul_abs=int((resul_freq*TotalR)/100)

            value={y:(resul_abs,resul_freq)}
            CoocurFreq.setdefault(x,value).update(value)

        # CoocurFreq -> { X : {'XIII': (1, 0.05), 'XII': (12, 0.39), 'XI':
(14, 0.46), 'XV': (0, 0.03)}, ... }
        return CoocurFreq


    #  _________________     _________________
    # |_________________| 5 |_________________|


    def ActualCoocur(RootsPats, TotalR):

      # RootsPats = {r: [p,p,p], ...}

        CoocurFreq={}
        for root,patterns in RootsPats.items():

PairsPats=list(itertools.combinations(sorted(patterns,key=DataExtraction.uti
lities_data.cmp_to_key(DataExtraction.utilities_data.numeric_compare)),2))

            # !! PairPats -> there are some cases in which a root has two
verbs of the same pattern; one of them is archaic
            for pair in PairsPats:
                x,y=pair[0],pair[1]
                if x in CoocurFreq:
                    if y in CoocurFreq[x]:
```

```
                    CoocurFreq[x][y]+=1
                else:
                    CoocurFreq[x][y]=1
            else: CoocurFreq[x]={y:1}

        for pat1,listafreq in CoocurFreq.items():
            for pat2,fq in listafreq.items():
                CoocurFreq[pat1][pat2]=(fq, round((fq*100)/TotalR,2))

        return CoocurFreq


    #  _________________     _________________
    # |_________________| 6 |_________________|


    def takes_radical_freqs_from_pats(code,lista_pats):
        'extracts a list of selected roots and gets the frequency data'
        list_target_roots=set()
        with open('lexicon_lemas_procesado.txt', encoding='utf8') as f:
            for line in f:
                try: l,r,c=line.strip().split()
                except: print(line)

                # ========= we define the variables to filter the roots we
want to extract the frequencies from =========== #
                       # length of root
                trilit = (code[0]=='1') and (len(r)==3)
                quadrilit = (code[0]=='2') and (len(r)==4)

                       # filter of geminated root
                GeminTri = (trilit) and ((code[1]=='2' and r[1]==r[2]) or
(code[1]=='1')) #and r[1]!=r[2]))
                GeminQua = (quadrilit) and ((code[1]=='2' and
r[0]+r[1]==r[2]+r[3]) or (code[1]=='1' and r[0]+r[1]!=r[2]+r[3]))

                       # filter of patterns
                matching_Pat = (code[2]=='1') or ((code[2]=='2') and (c in
lista_pats))
                #
================================================================================
============================== #

                if matching_Pat and (GeminTri or GeminQua):
                    list_target_roots.add(r)   # LIST OF TARGET ROOTS
```

```
        list_target_roots=list(list_target_roots)  # converts the set into a
list
        total_Roots=len(list_target_roots)         # total number of roots
        RadicalsFreq=[]                            # list to insert the
frequencies

        if len(list_target_roots[0])==3:
            length_root=3
            Radicals = [{},{},{}]      # triliteral roots
        else:
            length_root=4
            Radicals = [{},{},{},{}]   # quadriliteral roots

        for raiz in list_target_roots: # go through the list of roots
            i=0
            for r in raiz:     # takes each of the char from the root
                Radicals[i][r]=Radicals[i].get(r,0)+1
                i+=1
        Aux=[]
        for rad in Radicals:

Aux.append(DataExtraction.utilities_data.freq_dic(rad,total_Roots))
        RadicalsFreq.append(Aux)

        return RadicalsFreq, total_Roots, length_root



    def print_freqs(lista_freqR, total_R, length_root): # tri y qua
        print('\ttotal: %d\n' % (total_R))

        if length_root==3:
            print('Char\tR1abs\tR1%\tR2abs\tR2%\tR3abs\tR3%')
            for item in lista_freqR:
                R1,R2,R3=item[0],item[1],item[2]
                for cons,freq in R1.items():
                    r1abs,r1fq=freq[0],freq[1]

                    if cons in R2: r2abs,r2fq=R2[cons][0],R2[cons][1]
                    else: r2abs,r2fq=0,0

                    if cons in R3: r3abs,r3fq=R3[cons][0],R3[cons][1]
                    else: r3abs,r3fq=0,0

                    print(cons,r1abs,r1fq,r2abs,r2fq,r3abs,r3fq,sep='\t')
```

```
        elif length_root==4:
            print('Char\tR1abs\tR1%\tR2abs\tR2%\tR3abs\tR3%\tR4abs\tR4%')
            for item in lista_freqR:
                R1,R2,R3,R4=item[0],item[1],item[2],item[3]
                for cons,freq in R1.items():
                    r1abs,r1fq=freq[0],freq[1]

                    if cons in R2: r2abs,r2fq=R2[cons][0],R2[cons][1]
                    else: r2abs,r2fq=0,0

                    if cons in R3: r3abs,r3fq=R3[cons][0],R3[cons][1]
                    else: r3abs,r3fq=0,0

                    if cons in R4: r4abs,r4fq=R4[cons][0],R4[cons][1]
                    else: r4abs,r4fq=0,0


print(cons,r1abs,r1fq,r2abs,r2fq,r3abs,r3fq,r4abs,r4fq,sep='\t')

        return




    #  _________________     _________________
    # |_________________| 7 |_________________|


    def calculateBilitFreq(RootsPats, num_radicals):
        BilitFreq={}  # just patterns from biliteral root -> {pat: abs}

        if num_radicals=='3':
            for root,patterns in RootsPats.items():
                if root[1]==root[2]:
                    for pat in patterns:
                        BilitFreq[pat]=BilitFreq.get(pat,0)+1
            return BilitFreq

        elif num_radicals=='4':
            for root,patterns in RootsPats.items():
                if root[0]+root[1]==root[2]+root[3]:
                    for pat in patterns:
                        BilitFreq[pat]=BilitFreq.get(pat,0)+1
            return BilitFreq
```

```
    #  _________________          _________________
    # |_________________| 8, 9 |_________________|


    RootsPatsBoth=RootsPatsTri
    RootsPatsBoth.update(RootsPatsQua)


    def WithoutPatternI_OneRPatPerRoot(RootsPats):
        '''it does two things:
        extracts freq of patterns with one pattern per root
        and extracts freq of patterns without pattern I'''

        freqOnePat_per_root={}        # pattern per root Ratio = 1
        freqMultipleTri,freqTri={},{} # triliteral
        freqMultipleQua,freqQua={},{} # quadriliteral

        for Root,Patterns in RootsPats.items():
            Patterns=sorted(Patterns,
key=DataExtraction.utilities_data.cmp_to_key(DataExtraction.utilities_data.n
umeric_compare))
            Patterns=list(map((lambda p: re.sub('^I[aiu]{2}','I',p)),
Patterns))

            # freq pattern that meet pat/root=1
            if len(Patterns)==1:
                for pat in Patterns:

freqOnePat_per_root[pat]=freqOnePat_per_root.get(pat,0)+1

            # freq pattern from triliteral root with no form I
            if 'I' not in Patterns and len(Root)==3:
                # no. patterns from TriRoots without Form I
                for pat in Patterns:
                    freqTri[pat]=freqTri.get(pat,0)+1

                    # freq pattern combinations from TriRoots without
                    # Form I and more than one single pattern
                    if len(Patterns)>1:
                        pat=' - '.join(Patterns)
                        freqMultipleTri[pat]=freqMultipleTri.get(pat,0)+1

            # freq pattern from quadriliteral root with no form QI
            elif 'QI' not in Patterns and len(Root)==4:
                    # no. patterns from QuaRoots without Form QI
                    for pat in Patterns:
```

```
                    freqQua[pat]=freqQua.get(pat,0)+1

                # no. pattern combinations from QuaRoots without Form QI
                # and more than one single pattern
                if len(Patterns)>1:
                    pat=' - '.join(Patterns)
                    freqMultipleQua[pat]=freqMultipleQua.get(pat,0)+1

        return freqOnePat_per_root, freqTri, freqMultipleTri, freqQua,
freqMultipleQua


    #  _________________      _________________
    # |_________________| 10 |_________________|

    def freqVocalismOneGroup(RootsPats):

        '''Patterns             Vocalism

         Iau                     aa-au
         Iai,VII-XV,QIII,QIV     aa-ai
         Iuu                     au-au
         Iia                     ai-aa
         II,III,IV,QI            aa-ui
         Iaa,V,VI,QII            aa-aa
         Iii                     ai-ai'''

        FreqVoc={}
        for patterns in RootsPats.values():
            for pat in patterns:

                if pat=='Iau':
                    FreqVoc['aa-au']=FreqVoc.get('aa-au',0)+1

                elif pat in ['Iai','VII','VIII','IX','X','XI','XII',\
                             'XIII','XIV','XV','QIII','QIV']:
                    FreqVoc['aa-ai']=FreqVoc.get('aa-ai',0)+1

                elif pat=='Iuu':
                    FreqVoc['au-au']=FreqVoc.get('au-au',0)+1

                elif pat=='Iia':
                    FreqVoc['ai-aa']=FreqVoc.get('ai-aa',0)+1

                elif pat in ['II','III','IV','QI']:
                    FreqVoc['aa-ui']=FreqVoc.get('aa-ui',0)+1
```

```
                elif pat in ['Iaa','V','VI','QII']:
                    FreqVoc['aa-aa']=FreqVoc.get('aa-aa',0)+1

                elif pat=='Iii':
                    FreqVoc['ai-ai']=FreqVoc.get('ai-ai',0)+1

                else:
                    print('fail in pattern: %s' % pat)

        TotalVoc=0
        for i in FreqVoc.values(): TotalVoc=TotalVoc+i

        return FreqVoc, TotalVoc


    def freqVocalismSeparated(RootsPats):

        ''' Perfective

                Patterns
Vocalism
                  Iau,Iai,VII-XV,QIII,QIV,II,III,IV,QI,Iaa,V,VI,QII
aa
                  Iuu
au
                  Iia,Iii
ai

            Imperfective

                Patterns            Vocalism

                  Iau,Iuu                   au
                  Iai,VII-XV,QIII,QIV,Iii   ai
                  Iia,Iaa,V,VI,QII          aa
                  II,III,IV,QI              ui'''


        FreqVocP,FreqVocI={},{}
        for patterns in RootsPatsBoth.values():
            for pat in patterns:

                # perfective vocalism
                if pat in
['Iau','Iai','VII','VIII','IX','X','XI','XII','XIII','XIV',\
```

```
'XV','QIII','QIV','II','III','IV','QI','Iaa','V','VI','QII']:
                    FreqVocP['aa']=FreqVocP.get('aa',0)+1
                elif pat == 'Iuu':
                    FreqVocP['au']=FreqVocP.get('au',0)+1
                elif pat in  ['Iia','Iii']:
                    FreqVocP['ai']=FreqVocP.get('ai',0)+1
                else:
                    print('fail in pattern (perfective): %s' % pat)

                # imperfective vocalism
                if pat in ['Iau','Iuu']:
                    FreqVocI['au']=FreqVocI.get('au',0)+1
                elif pat in
['Iai','VII','VIII','IX','X','XI','XII','XIII','XIV','XV','QIII','QIV','Iii'
]:
                    FreqVocI['ai']=FreqVocI.get('ai',0)+1
                elif pat in ['Iia','Iaa','V','VI','QII']:
                    FreqVocI['aa']=FreqVocI.get('aa',0)+1
                elif pat in ['II','III','IV','QI']:
                    FreqVocI['ui']=FreqVocI.get('ui',0)+1
                else:
                    print('fail in pattern (imperfective): %s' % pat)

        TotalVocP,TotalVocI=0,0
        for i in FreqVocP.values(): TotalVocP=TotalVocP+i
        for i in FreqVocI.values(): TotalVocI=TotalVocI+i

        return FreqVocP, TotalVocP, FreqVocI, TotalVocI


    #  _________________      _________________
    # |_________________| 11 |_________________|


    def traditional_counting(VarForm):

        PerfectiveForms =
DataExtraction.utilities_data.saca_perfective_forms(VarForm)  # {pat: [form,
form, ...], ...}
        FreqProsody={}

        for pat,forms in PerfectiveForms.items():

            for f in forms:
                input_f = f                  # for checking errors in forms
```

```
                f=f.replace('ٱ','آ')        # madda normalization
                f=re.sub(r'(.)ٰ',r'\1ٰ\1',f)   # shadda normalization

                f = re.sub('.ْ','0',f)        # Convert SAKIN letter into 0
                for s in ['ى','ي','و','ا']:  # Convert MAMDOOD letter into 0
                    f = re.sub(s,'0',f)
                f = re.sub('.[ُِٰ]','1',f)      # Convert MUTAHARRIK letter
into 1

                if not re.search('[^10]',f):

                    # traditional accumulative counting conversion
                    f = re.sub('10','2',f)
                    f = re.sub('12','3',f)
                    f = re.sub('22','4',f)

                    # calculate total weight
                    n=sum(list(map(int,list(f))))

                    # convert into syllabic weight
                    f = re.sub('4','HH',f)   ## 1010 = 22 = 4 = HH
                    f = re.sub('3','LH',f)   ## 110  = 12 = 3 = LH
                    f = re.sub('2','H',f)    ## 10   = 10 = 2 = H
                    f = re.sub('1','L',f)    ## 1    = 1  = 1 = L

                    f = re.sub('H0','SH',f)  ## [H0 = SH] // SH computa lo
mismo que H

                else: print('Error in form: %s\tinput form: %s' % (f,
input_f))

                # pattern   total   forma_Prosody   freq_abs
                # {(pat,n,f):freq, ...}
                FreqProsody[(pat,n,f)]=FreqProsody.get((pat,n,f),0)+1

        return FreqProsody


    #
**************************************************************************
**

    salir = False
    while salir==False:
        option=input('''
```

```
    _____________________________________\n
        WRITE NUMBER OF SELECTED OPTION\n\n
        1\tnumber of roots, verbs and mean pattern per root\n
        2\tnumber patterns per root\n
        3\tfreq of patterns\n
        4\tpredicted (expected) freq of pattern co-occurrences\n
        5\tactual (observed) freq of pattern co-ocurrences\n
        6\tfreq of each radical from a specified list of patterns\n
        7\tfreq of patterns from triliteral roots that meet R2=R3
(biliterals)\n
        8\tfreq of each pattern for pat/root=1\n
        9\tfreq of patterns from roots without Form I\n
        10\tfreq of vocalism morphemes\n
        11\tfreq of patterns according to traditional counting of prosody\n
        0\texit\n
    _____________________________________\n''')


        if option == '0': salir=True


        elif option == '1':
            print('\nNUMBER OF ROOTS, VERBS AND MEAN PATTERNS PER ROOT')
            print('\ntriliteral roots: %d' % TotalRootsTri)
            print('triliteral verbs: %d' % TotalVerbsTri)
            print('pattern/triliteral root: %.2f\n' %
round(TotalVerbsTri/TotalRootsTri,2))

            print('quadriliteral roots: %d' % TotalRootsQua)
            print('quadriliteral verbs: %d' % TotalVerbsQua)
            print('pattern/quadriliteral root: %.2f\n' %
round(TotalVerbsQua/TotalRootsQua,2))


        elif option == '2':
            print('\n2. number patterns per root')
            pats_per_rootTri,pats_per_rootQua={},{}

            for i in Lemas_per_rootTri:
pats_per_rootTri[i]=pats_per_rootTri.get(i,0)+1  # triliteral
            print('\nPatterns\tNo. TriRoots\t%\n')
            for k,v in
DataExtraction.utilities_data.freq_dic(pats_per_rootTri,TotalRootsTri).items
():
                print(str(k).ljust(15),str(v[0]).ljust(15),v[1])
```

```
            for i in Lemas_per_rootQua:
pats_per_rootQua[i]=pats_per_rootQua.get(i,0)+1  # quadriliteral
            print('\n\nPatterns\tNo. QuaRoots\t%\n')
            for k,v in
DataExtraction.utilities_data.freq_dic(pats_per_rootQua,TotalRootsQua).items
():
                print(str(k).ljust(15),str(v[0]).ljust(15),v[1])


        elif option == '3':
            print('\nPat\tFreq\t% TriRoots\n')

DicKeys=sorted(list(freqPatTri.keys()),key=DataExtraction.utilities_data.cmp
_to_key(DataExtraction.utilities_data.numeric_compare))
            for p in DicKeys:

print(p.ljust(7),str(freqPatTri[p][0]).ljust(7),str(freqPatTri[p][1]))


            print('\n\nPat\tFreq\t% QuaRoots\n')

DicKeys=sorted(list(freqPatQua.keys()),key=DataExtraction.utilities_data.cmp
_to_key(DataExtraction.utilities_data.numeric_compare))
            for p in DicKeys:

print(p.ljust(7),str(freqPatQua[p][0]).ljust(7),str(freqPatQua[p][1]))


        elif option == '4':
            print('\npredicted freqs for Triliteral pattern co-
occurrences\n')
            PredCoocurFqTri=PredictCoocur(freqPatTri,TotalRootsTri)    #
gets predicted freqs
            DataExtraction.utilities_data.printDic_ordenado(PredCoocurFqTri)
# prints predicted freqs
            print('\npredicted freqs for Quadriliteral pattern co-
occurrences\n')
            PredCoocurFqQua=PredictCoocur(freqPatQua,TotalRootsQua)    #
gets predicted freqs
            DataExtraction.utilities_data.printDic_ordenado(PredCoocurFqQua)
# prints predicted freqs


        elif option == '5':
            print('\nactual freqs for Triliteral pattern co-occurrences\n')
```

```
            ActualCoocurFqTri=ActualCoocur(RootsPatsTri,TotalRootsTri)   #
gets predicted freqs

DataExtraction.utilities_data.printDic_ordenado(ActualCoocurFqTri)
# prints predicted freqs
            print('\nactual freqs for Quadriliteral pattern co-
occurrences\n')
            ActualCoocurFqQua=ActualCoocur(RootsPatsQua,TotalRootsQua)   #
gets predicted freqs

DataExtraction.utilities_data.printDic_ordenado(ActualCoocurFqQua)
# prints predicted freqs


        elif option == '6':
                # var_length:  tri=1         /  qua=2
                # var_gemin:   no_especif=1  /  yes=2
                # var_pat:     all=1         /  select=2  ->  var_pat_list:
[list of patterns]
                var_length=input('write 1 for triliteral roots or 2 for
quadriliteral:\n')
                var_gemin=input('write 1 all times of roots and 2 for
geminated roots:\n')
                var_pat=input('write 1 for all patterns or 2 if you want to
specify the patterns:\n')
                if var_pat.strip()=='2': var_list_pat=input('write the
pattern(s) separated by spaces:\n').strip().split()
                elif var_pat.strip()=='1': var_list_pat=[]
                            # 3-digit code with info: lenghth of root /
filter of patterns / filter of consonants
                code_filter=var_length+var_gemin+var_pat
                            # apply function that extracts the root list and
its freq
                radic_freqs, radic_total, length_r =
takes_radical_freqs_from_pats(code_filter,var_list_pat)
                            # prints the frequency data
                print_freqs(radic_freqs, radic_total, length_r)


        elif option == '7':
            print('\nPattern freq from triliteral roots that meet R2=R3\n')
            BilitFreq=calculateBilitFreq(RootsPatsTri,'3')

DicKeys=sorted(list(BilitFreq.keys()),key=DataExtraction.utilities_data.cmp_
to_key(DataExtraction.utilities_data.numeric_compare))
            for p in DicKeys:
```

```
                print(p,BilitFreq[p],sep='\t')

            print('\nPattern freq from quadriliteral roots that meet
R1+R2=R3+R4\n')
            BilitFreq=calculateBilitFreq(RootsPatsQua,'4')

DicKeys=sorted(list(BilitFreq.keys()),key=DataExtraction.utilities_data.cmp_
to_key(DataExtraction.utilities_data.numeric_compare))
            for p in DicKeys:
                print(p,BilitFreq[p],sep='\t')


        elif option == '8':
            print('\nfreq each pattern for pat/root=1\n')
            freqOnePat_per_root =
WithoutPatternI_OneRPatPerRoot(RootsPatsBoth)[0]
            for k,v in
DataExtraction.utilities_data.freq_dic(freqOnePat_per_root).items():
print(k.ljust(23),v)


        elif option == '9':

            freqOnePat_per_root, freqTri, freqMultipleTri, freqQua,
freqMultipleQua = WithoutPatternI_OneRPatPerRoot(RootsPatsBoth)

            print('\nfreq pats of roots without Form I\n')
            for k,v in
DataExtraction.utilities_data.freq_dic(freqTri).items():
print(k.ljust(23),v)

            print('\nfreq multiple patterns of roots without Form I for
pat/TriRoot>1\n')
            for k,v in
DataExtraction.utilities_data.freq_dic(freqMultipleTri).items():
print(k.ljust(34),v)

            print('\nfreq pats of roots without Form QI\n')
            for k,v in
DataExtraction.utilities_data.freq_dic(freqQua).items():
print(k.ljust(23),v)

            print('\nfreq multiple patterns of roots without Form QI for
pat/QuaRoot>1\n')
```

```
            for k,v in
DataExtraction.utilities_data.freq_dic(freqMultipleQua).items():
print(k.ljust(34),v)


        elif option == '10':

            VarVocalism = input('write 1 if you want to calculate the
frequencies for perfective and imperfective vocalism together, write 2 if
separatedly\n')

            if VarVocalism == '1':
                FreqVoc, TotalVoc = freqVocalismOneGroup(RootsPatsBoth)
                print('vocal\tlemas\tfreq\n')
                for k,v in FreqVoc.items():
                    freq=round(v*100/TotalVoc,1)
                    print(k,v,freq,sep='\t')
                print('\ntotal: %d' % TotalVoc)

            elif VarVocalism == '2':
                FreqVocP, TotalVocP, FreqVocI, TotalVocI =
freqVocalismSeparated(RootsPatsBoth)
                print('\nPerfective\nvocal\tlemas\tfreq\n')
                for k,v in FreqVocP.items():
                    freq=round(v*100/TotalVocP,1)
                    print(k,v,freq,sep='\t')
                print('\ntotal: %d' % TotalVocP)

                print('\n\nImperfective\nvocal\tlemas\tfreq\n')
                for k,v in FreqVocI.items():
                    freq=round(v*100/TotalVocI,1)
                    print(k,v,freq,sep='\t')
                print('\ntotal: %d' % TotalVocI)


        elif option == '11':
            FormToCount=input('Write 1 if you want to apply counting to
perfective form, and 2 for imperfective\n')
            FreqProsody = traditional_counting(FormToCount)  #
{(pat,n,f):freq, ...}

print('\n\nPattern'.ljust(11),'TotalWeight'.ljust(12),'SylStructure'.ljust(1
5),'freq')
            DicKeys=sorted(set([i[0] for i in
FreqProsody.keys()]),key=DataExtraction.utilities_data.cmp_to_key(DataExtrac
tion.utilities_data.numeric_compare))
```

```
            orden_previous=''  ## we store the previous pattern to know when
the pattern changes, so we can print a new line (for a clearer
visualization)
            for orden in DicKeys:
                if orden!=orden_previous: print('') # prints a new line
                orden_previous=orden
                for k,v in FreqProsody.items():
                    pat,total,syl,freq=k[0],k[1],k[2],v
                    if pat==orden:

print(pat.ljust(12),str(total).ljust(12),syl.ljust(12),freq)


    return ()
```